Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Работа допущена к защите

Директор ВШ КТиИС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Сушников

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

**Система ассистированного преобразования сигналов для задач с плохо формализуемыми метриками качества**

|  |  |  |
| --- | --- | --- |
| по направлению подготовки | |  |
| 09.04.01 Информатика и вычислительная техника | | |
| код и наименование направления подготовки (специальности) | | |
| направленность (профиль) |  | |
| 09.04.01\_15 Технологии проектирования системного и прикладного | | |
| код и наименование направленности (профиля) образовательной программы | | |
| программного обеспечения | | |

Выполнил

студент гр. 5140901/3150х Н.С.Васильев

Руководитель

доцент, к.т.н. К.К.Семёнов

Консультант по нормоконтролю

доцент, к.т.н. А.Г. Новопашенный

Санкт-Петербург

2025

Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Утверждаю

Директор ВШ КТиИС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Сушников

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

задание

на выполнение выпускной квалификационной работы магистерской диссертации

Студенту Васильеву Николаю Сергеевичу, группа 5140901/31501

1. Тема работы: «Система ассистированного преобразования сигналов для задач с плохо формализуемыми метриками качества»
2. Срок сдачи студентом законченной работы: 23.05.2025
3. Исходные данные по работе: нет
4. Содержание работы (перечень подлежащих разработке вопросов):
   1. Обзор предметной области, анализ аналогов и их сравнение (при наличии); обзор технологий.
   2. Выбор инструментов для разработки. Разработка технического задания.
   3. Разработка программного обеспечения
   4. Тестирование и апробация
   5. Практическое применение и исследование пользовательских свойств
5. Перечень графического материала (с указанием обязательных чертежей): Блок схемы алгоритмов, иллюстративный материал
6. Консультанты по работе: консультант по нормоконтролю – А.Г.Новопашенный
7. Дата выдачи задания 21.04.2025

Руководитель ВКР К.К.Семёнов

Задание принял к исполнению 21.04.2025

Студент Н.С. Васильев

**РЕФЕРАТ**

На 76 с., 14 рисунков, 2 таблицы, 3 приложения

КЛЮЧЕВЫЕ СЛОВА: некорректные задачи, пользовательские алгоритмы, интеллектуальные системы, визуальный анализ.

Тема выпускной квалификационной работы: «Система ассистированного преобразования сигналов для задач с плохо формализуемыми метриками качества».

Данная работа посвящена разработке веб-приложения, предназначенного для визуального анализа и обработки пользовательских одномерных данных при помощи настраиваемых Python-алгоритмов. В центре внимания находятся задачи, которые не поддаются формальному описанию метрик качества и требуют участия человека-эксперта.

Задачи, которые решались в ходе разработки:

1. Исследование особенностей некорректных задач и существующих программных решений в области обработки сигналов;
2. Проектирование архитектуры клиентского приложения с возможностью выполнения Python-кода на стороне пользователя;
3. Реализация интерфейса пошагового взаимодействия с системой и механизмов загрузки, параметризации и запуска пользовательских алгоритмов;
4. Обеспечение визуализации результатов и реактивного управления параметрами обработки.

В результате была разработана модульная веб-система, не требующая установки на устройство пользователя и реализующая сценарий пошагового взаимодействия: от загрузки данных и скрипта до визуального анализа результата. Система поддерживает интерполяционные, аналитические и пользовательские алгоритмы, выполняемые в изолированной среде браузера через Pyodide. Реализована адаптивная форма для задания параметров функций и интерактивный график, позволяющий оценить поведение алгоритма при различных настройках.

**ABSTRACT**

76 pages, 14 figures, 2 tables, 3 appendices

KEYWORDS: ill-posed problems, user adaptation, intelligent systems, visual analysis, human–machine interaction.

The subject of the graduate qualification work is «Assisted Signal Transformation System for Tasks with Poorly Formalized Quality Metrics».

The given work is devoted to the development of a web-based application designed for the visual analysis and processing of user-defined one-dimensional data using custom Python algorithms. The focus is on problems where the quality criteria cannot be explicitly formalized, and expert involvement is essential.

The work set the following goals:

1. To study the characteristics of ill-posed problems and review existing software tools in the field of signal processing;
2. To design a client-side application architecture that allows the execution of Python code directly in the browser;
3. To implement a step-by-step user interface for uploading data and algorithms, configuring parameters, and executing scripts;
4. To provide result visualization and reactive control over processing parameters (e.g., interpolation order).

The implementation resulted in a modular web-based system that requires no installation and supports an interactive workflow — from data upload and script parsing to result analysis. The system enables execution of interpolation, analysis, and user-defined algorithms within a secure browser-based environment using Pyodide. It includes a dynamic parameter input form and an interactive chart for real-time evaluation of algorithm behavior.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 9](#_Toc201219913)

[ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ 12](#_Toc201219914)

[1.1. Некорректные задачи 12](#_Toc201219915)

[1.1.1. Исторический контекст и теоретическая база 12](#_Toc201219916)

[1.1.2. Современная классификация некорректных задач 12](#_Toc201219917)

[1.1.3. Обоснование невозможности оптимизации 14](#_Toc201219918)

[1.1.4. Роль человека-эксперта в процессе 16](#_Toc201219919)

[1.2. Анализ аналогов и их сравнение 17](#_Toc201219920)

[1.2.1. MATLAB (Signal Processing Toolbox) 18](#_Toc201219921)

[1.2.2. OriginLab 19](#_Toc201219922)

[1.2.3. Chromeleon (CDS) 20](#_Toc201219923)

[1.2.4. Сравнение аналогов 22](#_Toc201219924)

[1.3. Обзор технологий 23](#_Toc201219925)

[1.3.1. Визуализация и пользовательский интерфейс 23](#_Toc201219926)

[1.3.2. Исполнение пользовательских алгоритмов 26](#_Toc201219927)

[1.3.3. Хранение пользовательского опыта и адаптация 29](#_Toc201219928)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 31](#_Toc201219929)

[2.1. Выбор инструментов разработки 31](#_Toc201219930)

[2.1.1. HTML 31](#_Toc201219931)

[2.1.2. SCSS (Sassy CSS) 31](#_Toc201219932)

[2.1.3. JavaScript 32](#_Toc201219933)

[2.1.4. Chart.js 32](#_Toc201219934)

[2.1.5. Pyodide 32](#_Toc201219935)

[2.2. Техническое задание 33](#_Toc201219936)

[2.2.1. Введение 33](#_Toc201219937)

[2.2.2. Основания для разработки 34](#_Toc201219938)

[2.2.3. Назначение разработки 34](#_Toc201219939)

[2.2.4. Требования к программе 35](#_Toc201219940)

[ГЛАВА 3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 36](#_Toc201219941)

[3.1. Структура веб-приложения 36](#_Toc201219942)

[3.1.1. Назначение и принципы построения 36](#_Toc201219943)

[3.1.2. Компоненты и структура проекта 38](#_Toc201219944)

[3.1.3. Поток данных и взаимодействие модулей 39](#_Toc201219945)

[3.2. Поддержка пользовательских данных и сценариев 40](#_Toc201219946)

[3.2.1. Типы поддерживаемых данных 40](#_Toc201219947)

[3.2.2. Обработка и валидация данных на клиенте 41](#_Toc201219948)

[3.2.3. Универсальность ввода 42](#_Toc201219949)

[3.3. Реализация поддержки Python-алгоритмов 43](#_Toc201219950)

[3.3.1. Обработка пользовательских скриптов и извлечение функций 43](#_Toc201219951)

[3.3.2. Универсальная модель вызова функции 44](#_Toc201219952)

[3.3.3. Поддержка аргументов и интерактивных примеров 45](#_Toc201219953)

[3.3.4. Изоляция среды исполнения и безопасность 46](#_Toc201219954)

[3.4. Обработка результата и визуальное представление 47](#_Toc201219955)

[3.4.1. Форматы выходных данных и поддерживаемые структуры 47](#_Toc201219956)

[3.4.2. Механизм передачи результата из Python в JavaScript 48](#_Toc201219957)

[3.4.3. Визуализация и масштабируемость 49](#_Toc201219958)

[3.5. Отчёт о реализации и оценка функциональности 50](#_Toc201219959)

[3.5.1. Поддерживаемые сценарии использования 50](#_Toc201219960)

[3.5.2. Ограничения текущей версии 51](#_Toc201219961)

[3.5.3. Направления развития 52](#_Toc201219962)

[3.6. Демонстрационный пример 53](#_Toc201219963)

[ГЛАВА 4. ТЕСТИРОВАНИЕ И АПРОБАЦИЯ 58](#_Toc201219964)

[4.1. Статический анализ 58](#_Toc201219965)

[4.1.1. Инструмент статического анализа 58](#_Toc201219966)

[4.1.2. Результаты проверки и типичные замечания 60](#_Toc201219967)

[4.2. Модульное тестирование 61](#_Toc201219968)

[4.2.1. Обоснование выбора инструмента 61](#_Toc201219969)

[4.2.2. Сценарии тестирования и проверка логики функций 61](#_Toc201219970)

[4.3. Динамический анализ 64](#_Toc201219971)

[4.3.1. Цели динамического анализа и выбранный инструмент 64](#_Toc201219972)

[4.3.2. Сценарии профилирования и результаты 65](#_Toc201219973)

[4.4. Выводы по результатам тестирования и анализа 70](#_Toc201219974)

[ЗАКЛЮЧЕНИЕ 72](#_Toc201219975)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 74](#_Toc201219976)

[Приложение 1 77](#_Toc201219977)

[Приложение 2 115](#_Toc201219978)

[Приложение 3 118](#_Toc201219979)

# ВВЕДЕНИЕ

Существует большое количество задач, в которых невозможно или затруднительно задать количественный критерий оценки результата, пригодный для численной оптимизации. К таким задачам относятся, например, разделение пересекающихся хроматографических пиков, восстановление пропусков в данных, обратная фильтрация и другие. Общей особенностью таких задач является необходимость экспертной оценки результата на основе визуального анализа или интуитивного представления о правильности решения.

В настоящее время подобные задачи решаются преимущественно с использованием заранее определённых алгоритмов, параметры которых подбираются вручную. На практике это приводит к необходимости многократного запуска пользовательских программ с различными конфигурациями и субъективной оценки результата, что значительно снижает эффективность работы. Распространённые программные средства, как правило, не предоставляют средств для интуитивной настройки параметров в реальном времени. Остаются неразрешёнными вопросы создания универсального инструмента для взаимодействия с пользовательскими алгоритмами, организации вычислений с высокой производительностью при значительных объёмах данных, а также накопления и использования экспертного опыта в процессе настройки. Эти аспекты определяют необходимость разработки интерактивного инструмента, ориентированного на помощь пользователю в решении задач, в которых отсутствует формализованный подход к оценке результата.

Объектом исследования являются прикладные задачи обработки сигналов, в которых результат оценивается преимущественно визуально. Предметом исследования являются методы и программные средства, обеспечивающие интерактивную настройку параметров пользовательских алгоритмов в условиях отсутствия объективной метрики.

Целью исследования является разработка системы-помощника, которая обеспечивает ассистированную настройку параметров пользовательских алгоритмов обработки сигналов в задачах, где оценка качества результата затруднена и осуществляется экспертом на основе визуального анализа и интуитивного представления о корректности решения.

В соответствии с поставленной целью были сформированы следующие задачи.

1. Изучить особенности прикладных задач обработки сигналов с экспертной визуальной оценкой качества результата, а также проанализировать существующие подходы и программные решения, применяемые в данной области.
2. Выполнить обзор технологий и инструментов для реализации системы, а также сформулировать техническое задание на её разработку.
3. Разработать архитектуру программной системы, ориентированной на визуальную настройку параметров пользовательских алгоритмов.
4. Реализовать программную систему, обеспечивающую загрузку пользовательских скриптов обработки данных, интерактивную настройку параметров и накопление истории взаимодействий.
5. Провести тестирование и апробацию системы на типовых примерах задач.

Если внедрить механизм интерактивного управления параметрами, влияющими на результат задачи, и тем самым избавиться от необходимости многократного ручного перезапуска программы обработки с различными значениями этих параметров, то эффективность решения задач, не имеющих формализуемой оценки качества, повысится за счёт упрощения процесса подбора значений, обеспечивающих удовлетворительный для пользователя результат.

Если система будет накапливать и учитывать историю взаимодействия пользователя при работе с однотипными задачами, то это позволит адаптировать поведение системы и сократить время, необходимое для достижения подходящего результата.

При подготовке ВКР были использованы материалы учебной дисциплины «Технологии разработки программного обеспечения».

В ходе научно-исследовательской работы и преддипломной практики были изучены типовые прикладные задачи обработки сигналов без формализуемой метрики качества, а также реализован и апробирован прототип системы для интерактивной настройки параметров пользовательских алгоритмов. Выполнение алгоритмов осуществлялось на стороне клиента, а тестирование проводилось на задаче обратной фильтрации изображения.

Причины, повлиявшие на выбор темы, связаны с растущей потребностью в автоматизированных инструментах поддержки принятия решений в задачах, не обладающих чётко определёнными критериями оптимальности. В современных научных и инженерных практиках наблюдается смещение фокуса с формальных алгоритмов на гибкие, адаптивные системы, ориентированные на взаимодействие с пользователем.

Научная новизна работы заключается в разработке и апробации подхода, в рамках которого пользователь участвует в процессе обработки данных через интерактивную систему, предоставляющую возможность выбора алгоритмов, задания параметров и визуальной оценки результата.

Разработанное решение позволяет повысить удобство и эффективность работы с задачами, в которых результат оценивается на основе экспертного восприятия. Полученные наработки могут применяться в прикладных исследованиях.

Результаты, полученные в ходе исследования, были представлены в виде научного доклада на «Неделе науки» института компьютерных наук и кибербезопасности Санкт-Петербургского политехнического университета Петра Великого.

# ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

## Некорректные задачи

### Исторический контекст и теоретическая база

Понятие корректности задачи было сформулировано в трудах французского математика Жака Адамара в начале 20-го века. В своих работах он выявил три необходимых условия корректности задачи:

1. решение задачи должно существовать.
2. решение должно быть единственным.
3. решение должно непрерывно зависеть от входных данных.

Если хотя бы одно из этих условий нарушается, то задача считается некорректной.

С появлением вычислительной техники стало возможным численно решать некорректные задачи с использованием приближённых, устойчивых методов [1]. Тем не менее, при их широком распространении в прикладных областях, таких как медицинская диагностика (томография), геофизика (интерпретация сейсмических данных), аналитическая химия (хроматография, спектроскопия), восстановление изображений и сигналов, оказалось, что часто корректность в математическом смысле недостижима и полноценное решение должно опираться не только на формальные критерии, но и на экспертную оценку [2].

Некорректные задачи представляют собой не исключение, а важный и широко распространённый класс проблем, требующих гибких, адаптивных и человеко-ориентированных подходов к их решению. Их теоретическая база опирается на идеи устойчивости, регуляризации и эвристической интерпретации результатов, что делает необходимым сочетание методов с интерактивными средствами анализа и экспертного контроля.

### Современная классификация некорректных задач

В современной научной литературе некорректные задачи классифицируются по различным основаниям в зависимости от причин их некорректности, особенностей постановки, а также области применения. Несмотря на разнообразие формулировок, выделяют несколько устойчивых классов, каждый из которых характеризуется собственными типовыми трудностями и подходами к решению.

1. Обратные задачи (inverse problems)

Это наиболее типичный и изученный класс некорректных задач. Они связаны с восстановлением скрытых причин по наблюдаемым следствиям [25]. К этому классу относятся задачи восстановления изображения, восстановления распределения плотности по данным томографического сканирования, определения внутренней структуры по поверхности волнового поля. В таких задачах нарушается условие устойчивости: малые изменения во входных данных могут приводить к резким изменениям в решении [36].

1. Задачи восстановления или реконструкции (reconstruction problems)

К этому классу относят те задачи, в которых требуется заполнить недостающую информацию: интерполяция пропусков в сигнале, восстановление утраченных значений в таблице [32]. При отсутствии априорной информации множество возможных решений может быть бесконечным и выбор одного из них невозможен без привлечения экспертных оценок. Такие задачи не имеют единственного корректного ответа.

1. Задачи деконволюции и фильтрации (deconvolution, filtering)

К этому классу относится, например, задача обратной фильтрации, при которой по искажённому сигналу необходимо восстановить исходный. Часто при этом применяются аппроксимационные методы, однако без чёткого критерия качества невозможно автоматизировать выбор параметров фильтра.

1. Задачи разделения компонент (source separation, deblending)

К данному классу относятся разделение пересекающихся хроматографических пиков, звуковых сигналов, наложенных изображений. Результат зависит от предположений о форме, числе и структуре составляющих компонент. В этих задачах некорректность проявляется в неоднозначности интерпретации: возможно множество решений, соответствующих наблюдаемым данным, но различающихся по содержанию.

1. Задачи кластеризации и упрощённой интерпретации (clustering, abstraction)

Хотя формально данные задачи не всегда считаются некорректными, в ряде случаев выбор числа кластеров или параметров алгоритма приводит к неустойчивым и неоднозначным результатам. Такие задачи могут рассматриваться как плохо обусловленные с точки зрения интерпретации.

Все перечисленные классы объединяет ключевая особенность: решение не определяется однозначно и не может быть автоматически выбрано с помощью универсальной количественной метрики. Это делает невозможным применение прямых методов оптимизации и требует привлечения экспертного мнения, интерактивной настройки и визуального контроля.

Современная классификация некорректных задач учитывает характер априорных ограничений:

* наличие априорной информации (регуляризаторы, ограничения);
* свойства искомого решения (гладкость, разреженность, локальность);
* требования к устойчивости и интерпретируемости.

В зависимости от этого применяются различные подходы к решению: методы регуляризации (Тихонова, энтропийные) [31], байесовские подходы, обучаемые модели и человеко-машинное взаимодействие, где система играет вспомогательную роль, а окончательно решение принимается человеком на основе визуального анализа и опыта.

Понимание структуры и особенностей некорректных задач позволяет выбирать подходящие способы их практического решения – в том числе с использованием интерактивных ассистентов.

### Обоснование невозможности оптимизации

Одним из главных затруднений при решении некорректных задач в прикладных областях является отсутствие формализуемой функции качества, которая могла бы быть использована в качестве критерия для автоматического подбора параметров обработки. В большинстве алгоритмических подходов оптимизация опирается на количественно определённую целевую функцию (метрика отклонения, функция потерь). Однако в ряде задач, особенно в области визуальной аналитики и обработки экспериментальных данных, качество результата оценивается исключительно по субъективным признакам, таким как форма сигнала, наличие артефактов, степень разделения компонент.

Ниже представлены типовые примеры, где автоматизация оптимизации оказывается невозможной или неэффективной.

* В задаче разделения пересекающихся хроматографических пиков существует множество допустимых конфигураций параметров, которые формально дают допустимое разложение сигнала. Лишь некоторые из них интерпретируются экспертом как физически обоснованные, в зависимости от контекста анализа [34].
* В задаче обратной фильтрации изображения использование классических оптимизируемых функций, таких как среднеквадратичное отклонение, может приводить к избыточному усилению шума или искажению деталей. При этом визуально более приемлемый результат может иметь более высокое значение ошибки по формальной метрике. Возникает конфликт между формальным минимумом функции и восприятием качества человеком [35].

Невозможность задать универсальную метрику качества решения делает недостижимой традиционную процедуру численной оптимизации. Любой автоматический перебор параметров без учёта субъективной оценки оказывается либо неэффективным, либо приводит к результатам, не соответствующим требованиям специалиста.

Эту особенность обозначают как неформализуемость критерия качества. В таких условиях наиболее разумным подходом становится вовлечение пользователя в контур обратной связи, при котором система обеспечивает быструю реакцию на изменение параметров, а окончательное решение выбирается экспертом по визуальному или интуитивному признаку.

Применение методов оптимизации оказывается невозможным по самой природе задачи. Это требует перехода от классической автоматизированной обработки к ассистированному взаимодействию, что и лежит в основе разрабатываемого решения в рамках данной ВКР.

### Роль человека-эксперта в процессе

В условиях отсутствия формализуемого критерия качества решения ключевую роль в выборе подходящего результата играет эксперт, чьи знания, опыт и интуиция становятся основой для интерпретации полученных данных. При этом эксперт не просто оценивает финальный результат – он становится активным участником вычислительного процесса, формируя обратную связь на каждом этапе подбора параметров обработки.

Такое взаимодействие относится к области человеко-машинного сотрудничества в рамках интеллектуальных систем, где пользователь выполняет функции, недоступные формальным алгоритмам: принимает решения в условиях неопределённости, интерпретирует неоднозначные данные, использует априорные знания, не поддающиеся формализации. В этом контексте система выступает не как автономный исполнитель, а как ассистент, помогающий пользователю быстрее находить удовлетворительное решение.

Во многих прикладных областях визуальная интерпретация результата оказывается более надёжным ориентиром, чем численная метрика. В таких случаях даже небольшое расхождение с субъективно ожидаемым результатом делает формально «оптимальное» решение непригодным. Поэтому в задачах разделения сигналов, фильтрации изображений или восстановления данных именно человек определяет, насколько результат можно считать приемлемым, и направляет процесс его поиска.

В связи с этим всё более востребованными становятся полуавтоматические подходы, сочетающие вычислительные возможности машины с гибкостью и интуитивной оценкой человека. Такой подход позволяет сохранить контроль над процессом в руках эксперта, одновременно ускоряя и упрощая рутинные действия, такие как настройка параметров или повторный запуск обработки. Особенно важным это становится при многократной работе с однотипными задачами, когда накапливается опыт пользователя, который может быть использован для адаптации поведения системы.

Таким образом, в задачах, в которых невозможно формально задать количественный критерий качества, участие человека не просто необходимо – оно встроено в контур обработки, и эффективность решения напрямую зависит от того, насколько удобно и эффективно организовано взаимодействие пользователя с системой. Это обосновывает необходимость разработки ассистирующего программного решения, предоставляющего интерфейс для наглядной настройки, мгновенной обратной связи и последующего учёта предпочтений пользователя. Полуавтоматическая архитектура оказывается логичным и прагматичным выборов в условиях, где полная автоматизация невозможна.

## Анализ аналогов и их сравнение

Для решения задач обработки сигналов, изображений и других экспериментальных данных существует широкий спектр программных средств, разработанных как для научных, так и для прикладных целей. Многие из них активно применяются в таких областях, как физика, химия, биоинформатика, медицина, инженерия. Однако практически все эти системы разрабатывались с прицелом на задачи, обладающие чётко заданной формулировкой, численной метрикой качества результата и возможностью автоматической оптимизации. В условиях, когда задача не удовлетворяет условиям корректности по Адамару, а её решение требует субъективной экспертной оценки, функциональные возможности стандартных программных платформ оказываются ограниченными.

В данной подглаве рассмотрен ряд наиболее популярных и широко применяемых систем для данных задач, в числе которых – MATLAB, OriginLab и Chromeleon. Анализ проводился с позиций пяти критериев, важных для задач с плохо формализуемой метрикой качества: возможности подключения пользовательских алгоритмов, возможности визуальной настройки параметров, способности к накоплению пользовательского опыта и присутствия ориентации на экспертную интерпретацию результата.

### MATLAB (Signal Processing Toolbox)

MATLAB [23] – одна из самых распространённых программных сред для вычислений, активно применяемая как в академической, так и в промышленной среде. Система, разработанная компанией MathWorks, предоставляет широкие возможности для анализа и синтеза сигналов, моделирования систем, численного интегрирования, фильтрации и визуализации данных. Модуль Signal Processing Toolbox является одним из наиболее распространённых компонентов MATLAB и включает обширный набор функций для работы с временными рядами, спектрами, фильтрами, статистикой сигналов и их преобразованиями.

Сильной стороной MATLAB является его гибкость. Пользователи могут писать собственные скрипты, функции и приложения с использованием как встроенных средств визуализации, так и графических интерфейсов. Это делает платформу мощным инструментом в задачах, где необходима точная настройка и экспериментирование с алгоритмами обработки. MATLAB поддерживает разнообразные форматы входных данных, взаимодействие с внешними устройствами, а также интеграцию с другими языками программирования, такими как Python, C++ и Java.

Однако несмотря на все эти преимущества, при решении задач с неформализуемой метрикой качества пользователь может столкнуться с рядом ограничений.

* Интерфейс ориентирован на скриптовую модель взаимодействия, в которой пользователь задаёт параметры вручную, запускает скрипт и ожидает результата. Визуальное управление параметрами реализуется только при создании специализированного пользовательского интерфейса, что требует дополнительной разработки.
* MATLAB не предоставляет встроенных средств для накопления истории пользовательской активности или обучения на предыдущем опыте взаимодействия. Каждый запуск обработки воспринимается как изолированное событие и система не делает попыток адаптироваться к предпочтениям пользователя.
* Платформа ориентирована на формальные методы обработки, оптимизируемые по определённой метрике. В задачах, где результат должен оцениваться визуально, MATLAB не предоставляет встроенных механизмов поддержки экспертного выбора.

Несмотря на богатый функционал и высокую степень гибкости, MATLAB остаётся средой, ориентированной на задачи с чёткой математической постановкой и ограниченной поддержкой задач, требующих субъективно оцениваемой обработки.

### OriginLab

OriginLab [14] – это программный продукт для визуализации, обработки и анализа данных, широко используемый в научных и инженерных исследованиях. Он представляет собой удобный инструмент для представления экспериментальных данных в виде графиков и выполнения статистического анализа, аппроксимации, анализа кривых и других операций. В отличие от MATLAB, OriginLab ориентирован прежде всего на графический интерфейс пользователя, минимизирует необходимость программирования со стороны пользователя и обеспечивает доступ к функциям обработки через интерактивные диалоговые окна и мастер-настройки.

Одной из ключевых особенностей OriginLab является встроенная поддержка анализа пиков – функция, активно применяемая в хроматографии, спектроскопии и других смежных областях. Пользователь может определять пороговые значения, выполнять вычитание фона, строить аппроксимационные модели и выделять параметры пиков. Однако, несмотря на кажущуюся интерактивность, управление параметрами обработки остаётся пошаговым и дискретным. Каждое изменение значения параметра требует закрытия диалогового окна и последующего перерасчёта. Это препятствует интуитивной и быстрой настройке.

Система поддерживает ограниченное подключение пользовательских алгоритмов. С помощью встроенного языка программирования LabTalk и более современного Python API пользователь может создавать макросы, скрипты и пользовательские функции, но внедрение новых методов требует изучения архитектуры OriginLab и зачастую ограничено рамками предоставляемых интерфейсов. Интеграция собственных обработчиков сигналов или сложных вычислительных моделей, как правило, осуществляется вне визуального интерфейса и требует программных обходных решений.

Как и в случае с MATLAB, система не ведёт истории пользовательских действий и не предлагает адаптацию поведения под конкретного пользователя. Каждая сессия анализа начинается без учёта предыдущих успешных конфигураций.

OriginLab ориентирован на объективную аппроксимацию данных по числовым критериям (среднеквадратичное отклонение, коэффициент корреляции, качество подгонки). В задачах, где оценка качества результата не поддаётся формализации и осуществляется экспертом, система предоставляет лишь ограниченные возможности для гибкой настройки.

Таким образом, OriginLab представляет собой мощный инструмент для научной визуализации и анализа, но его возможности существенно ограничены в контексте задач, требующих интерактивного управления параметрами, немедленной визуальной обратной связи и учёта экспертной оценки результата без опоры на формализованные критерии.

### Chromeleon (CDS)

Chromeleon [29] – это программная платформа класса Chromatography Data System (CDS), разработанная компанией Thermo Fisher Scientific для управления хроматографическим оборудованием и обработки получаемых данных.

Среди функциональных возможностей Chromeleon – сбор данных с приборов, автоматическое выделение пиков, интеграция, количественный анализ, формирование отчётов и валидация на соответствие стандартам. Интерфейс системы ориентирован на стандартизированные, повторяющиеся рабочие процессы, которые требуют минимального вмешательства пользователя и обеспечивают высокую степень точности.

Несмотря на наличие визуальных инструментов анализа, Chromeleon строго ограничен своей предметной областью – хроматографией. Он не предназначен для решения более широкого круга задач обработки сигналов и не предоставляет средств для настройки произвольной логики обработки. Пользователь может варьировать параметры встроенных алгоритмов, однако использование пользовательских алгоритмов, нестандартных вычислительных схем или сторонних скриптов невозможно.

Взаимодействие с параметрами обработки в Chromeleon также осуществляется в пошаговом режиме. Любое изменение требует ручного подтверждения и последующего перерасчёта. Визуальная обратная связь не является реактивной, что ограничивает возможность интуитивного и гибкого подбора конфигурации.

Chromeleon не реализует механизмы адаптации к пользователю. История действий не сохраняется в целях персонализации и поведение системы не меняется в зависимости от контекста задачи.

Таким образом, Chromeleon является очень узкоспециализированным аналогом разработанной системы. Он представляет интерес как пример существующей визуальной среды, ориентированной на экспертную интерпретацию, но его жёсткая специализация на хроматографических задачах, закрытость архитектуры и отсутствие универсальности делают его неспособным решать широкий спектр задач с плохо формализуемыми метриками качества.

### Сравнение аналогов

Ниже приведена таблица сравнения аналогов по ключевым критериям (табл. 1.1). Из представленного сравнения видно, что ни одна из существующих систем не удовлетворяет всем требованиям, предназначенным для работы с задачами без формализуемой метрики качества результата.

Таблица 1.1

Сравнение аналогов системы по ключевым критериям

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | MATLAB (Signal Processing Toolbox) [23] | OriginLab [14] | Chromeleon (CDS) [29] |
| Возможность подключения пользовательских алгоритмов | Да. Через скрипты и собственные функции. | Ограниченно. Через встроенные языки (LabTalk, Python API) | Нет. Поддерживаются только встроенные алгоритмы. |
| Визуальная настройка параметров | Частично. Возможна при самостоятельной реализации GUI. | Да. Через интерфейс, но с пошаговым подтверждением. | Да, но строго в рамках встроенных параметров. |
| Накопление пользовательского опыта | Нет. Сессии не запоминаются. | Нет. История конфигураций не сохраняется. | Нет. Работа организована по фиксированным шаблонам. |
| Ориентация на экспертную интерпретацию результата | Нет. Система ориентирована на формальные метрики. | Частично. Визуализация помогает, но основной упор на численные оценки. | Частично. Есть визуальные инструменты, но имеется жёсткое ограничение предметной областью. |

Таким образом, MATLAB является мощной средой с возможностью подключения пользовательских алгоритмов, но требует самостоятельной реализации визуального интерфейса и не предоставляет реактивности и адаптивности. OriginLab предоставляет готовые визуальные инструменты, но не поддерживает произвольную логику обработки и не подходит для пошаговой настройки с немедленным откликом. Chromeleon строго ограничен и узкоспециализирован на хроматографии, закрыт для пользовательской логики и ориентирован на регламентированную обработку, а не на исследовательскую настройку.

Эти ограничения подчёркивают актуальность разработки специализированной системы, в которой визуальная, реактивная, настраиваемая среда будет изначально заложена в архитектуру.

## Обзор технологий

Разработка интерактивной системы для ассистированного преобразования сигналов для задач с плохо формализуемыми метриками качества требует использования современного технологического стека, обеспечивающего одновременно:

* исполнение пользовательских алгоритмов обработки;
* визуальное управление параметрами в интерфейсе;
* немедленную обратную связь на изменение параметров;
* расширяемость и адаптируемость;
* разделение вычислительного и пользовательского уровней.

В данной подглаве рассмотрены технологии, которые применяются для реализации таких систем: от средств визуализации и управления до механизмов выполнения кода и организации архитектуры.

### Визуализация и пользовательский интерфейс

Одной из ключевых особенностей разрабатываемой системы является возможность визуального управления параметрами обработки данных и наглядной оценки результата. Это требует реализации интерфейса, обеспечивающего удобную и интуитивную визуализацию сигналов, а также поддержку динамического взаимодействия между пользователем и системой. Для этого использованы современные веб-технологии и библиотеки, позволяющие разрабатывать интерактивные пользовательские интерфейсы с высокой отзывчивостью:

1. HTML, CSS, JavaScript

Базисом для любой веб-интерфейсной реализации являются HTML (HyperText Markup Language), CSS (Cascading Style Sheets) и JavaScript. HTML [8] используется для разметки структуры страниц: размещения графиков, управляющих элементов (кнопок, слайдеров, полей ввода и т.д.). CSS [4] – для оформления интерфейса, создания адаптивных и читаемых визуальных блоков, настройки отступов, цветов, размеров и шрифтов. JavaScript обеспечивает логику взаимодействия на клиентской стороне: обработку событий, управление данными и обновление элементов DOM без перезагрузки страницы. Эти три технологии являются основой любого веб-приложения и позволяют создавать независимый от операционной системы интерфейс, доступный через браузер.

1. React

React [18] – это современная JavaScript-библиотека для построения пользовательских интерфейсов, разработанная Facebook. Она реализует реактивную модель взаимодействия, при которой изменение состояния (например, перемещение ползунка) автоматически вызывает обновление визуального представления данных. React особенно удобен для реализации сложных, многослойных интерфейсов благодаря:

* компонентной архитектуре (весь интерфейс разбивается на переиспользуемые модули);
* виртуальному DOM, обеспечивающему высокую производительность;
* поддержке одностороннего потока данных, что упрощает отладку и предсказуемость поведения;
* возможности использования хуков и состояний, благодаря которым параметры визуализации и обработки могут быть синхронизированы с текущим состоянием приложения.

В контексте разработанной системы React позволяет создать интерфейс, в котором:

* пользователь может выбирать и настраивать параметры обработки с помощью ползунков и других интерактивных элементов;
* изменения параметров немедленно отражаются в визуализации сигнала;
* все изменения происходят без перезагрузки страницы, что обеспечивает непрерывность взаимодействия.

1. Chart.js

Chart.js [3] – это библиотека на JavaScript для построения графиков, основанная на элементе canvas HTML5. Она используется для визуального отображения сигналов, графиков зависимости параметров, результатов обработки и других данных в разрабатываемой системе.

К преимуществам chart.js может быть отнесено следующее.

* Простой и понятный API для быстрого построения линейных графиков, гистограмм, круговых диаграмм и др.
* Поддержка динамического обновления данных: можно перерисовывать график в реальном времени при изменении входных значений.
* Гибкие возможности настройки внешнего вида (цвета, подписи, сетка, масштаб).
* Поддержка масштабирования, наведения, всплывающих подсказок.

Для задач визуального анализа сигналов Chart.js подходит благодаря:

* высокой производительности при отрисовке одномерных данных;
* лёгкости интеграции с React и другими фреймворками;
* возможности быстро «перерисовывать» результат при каждом изменении параметра, обеспечивая реактивную визуальную обратную связь.

1. Альтернативные библиотеки (Plotly.js, D3.js)

Хотя в разработке может использоваться Chart.js, стоит упомянуть и другие библиотеки для визуализации данных.

* Plotly.js [16] – мощная библиотека для сложных графиков, включая 3D, тепловые карты, спектры и интерактивные отчёты. Подходит для научных и инженерных данных.
* D3.js [5] – низкоуровневая библиотека для создания полностью кастомизированных визуализаций. Требует более глубоких знаний, но предоставляет максимальную гибкость.

Для целей быстрого, отзывчивого и стабильного отображения одномерных сигналов Chart.js является наиболее удобным решением из представленных.

### Исполнение пользовательских алгоритмов

Одной из принципиальных особенностей разрабатываемой системы является возможность использовать пользовательские скрипты обработки сигналов, реализованные на языке Python. Это означает, что пользователь не просто настраивает параметры готового алгоритма, а загружает собственный код, который должен быть выполнен системой и отдать результат обработки в виде визуализации.

Такой подход требует наличия технологии, обеспечивающей безопасное и изолированное исполнение кода, поддержку ключевых научных библиотек (NumPy, SciPy), взаимодействие между пользовательским интерфейсом и средой выполнения, кроссплатформенность и независимость от серверной инфраструктуры (на стадии прототипа).

Рассмотрим наиболее популярные решения.

1. Pyodide

Pyodide [17] – это проект, разработанный Mozilla, представляющий собой сборку интерпретатора Python, переведённого в WebAssembly [24] и работающего прямо в браузере без сервера [17]. К ключевым особенностям относят:

* полноценный Python 3.11, запускаемый на клиентской стороне;
* поддержку популярных библиотек: NumPy [13], Pandas [15], Matplotlib [11], SciPy [22], scikit-learn [21] и другие;
* возможность взаимодействия между Python и JavaScript – можно передавать данные в обе стороны (например, передавать параметры из React в Pyodide и обратно);
* работа полностью изолирована в браузере, что обеспечивает безопасность исполнения и отсутствие зависимости от серверной архитектуры.

В рамках прототипа системы Pyodide идеально подходит для выполнения пользовательских скриптов без необходимости настройки серверной среды, обеспечения немедленного отклика, так как исполнение происходит локально в оффлайн-режиме (при наличии загруженных библиотек). К ограничениям относят производительность, количество доступных библиотек и большой размер инициализации.

1. Серверное исполнение кода (Python + API)

Альтернативой клиентскому исполнению является выполнение пользовательских алгоритмов на сервере, что подходит для более тяжёлых или длительных вычислений. В этой архитектуре пользовательский скрипт и параметры отправляются на сервер, где запускаются в изолированной среде (например, в Docker-контейнере), а затем результат возвращается на клиент. Пример возможной реализации клиент-серверной архитектуры разработанной системы представлен на Рис. 1.1.

Изображение выглядит как текст, снимок экрана, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Возможная реализация архитектуры разработанной системы

Такой подход позволяет использовать любые библиотеки Python без ограничений, выполнять ресурсоёмкие алгоритмы с высокой производительностью, масштабировать систему для одновременной работы нескольких пользователей.

Сложность заключается в необходимости инфраструктуры: требуется сервер, безопасная среда выполнения, средства асинхронной обработки задач и механизмы коммуникации с клиентом (например, через REST API [19]). Это увеличивает объём поддержки и снижает интерактивность в условиях нестабильного соединения.

Для наглядности приведена таблица сравнения двух подходов (Табл. 1.2).

Таблица 1.2

Сравнительная таблица подходов к исполнению пользовательских алгоритмов

|  |  |  |
| --- | --- | --- |
| Характеристика | Pyodide (исполнение в браузере) | Серверное исполнение |
| 1 | 2 | 3 |
| Место выполнения | На клиенте (в браузере) | На сервере (удалённо) |
| Поддержка библиотек | Ограниченная (только встроенные) | Полная |

Продолжение табл. 1.2

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Производительность | Средняя | Высокая |
| Безопасность | Высокая, изолировано в браузере | Требуется контейнеризация |
| Необходимость инфраструктуры | Не требуется | Требуется сервер и API |
| Скорость отклика | Максимальная | Зависит от канала связи |
| Подходит для прототипа | Да | Частично |
| Масштабируемость | Ограниченная | Высокая |

Для целей начальной реализации и прототипирования оптимальным решением является использование Pyodide, поскольку он позволяет достичь высокой интерактивности без зависимости от серверной части. Такой подход обеспечивает мгновенное выполнение пользовательских алгоритмов, визуальный отклик и простоту развёртывания.

На этапах, связанных с повышенными нагрузками, расширением вычислительных возможностей или многопользовательским режимом, целесообразен переход к серверной архитектуре, сочетающей гибкость, производительность и масштабируемость.

### Хранение пользовательского опыта и адаптация

Одной из ключевых отличительных особенностей разрабатываемой системы будет являться её способность накапливать информацию о действиях пользователя и адаптировать поведение интерфейса и параметров обработки на основе ранее успешных взаимодействий. Такая функциональность выводит систему за рамки стандартных инструментов визуального анализа и делает её полуавтоматическим интеллектуальным ассистентом, способным подстраиваться под стиль работы конкретного пользователя.

При работе с задачами с плохо формализуемыми метриками качества пользователь обычно подбирает параметры эмпирически, путём последовательного изменения значений и визуальной оценки результата. Этот процесс нередко сопровождается повторяющимися действиями: пользователь возвращается к тем же диапазонам значений, повторяет комбинации, которые ранее приводили к приемлемому результату.

Накопление этой информации и использование её в последующих сессиях позволяет:

* сокращать число итераций настройки;
* предлагать начальные значения параметров, близкие к удачным;
* выявлять неэффективные конфигурации и исключать их;
* адаптировать интерфейс под наиболее востребованные функции.

Для реализации механизма адаптации необходимо сохранять историю параметров, применяемых пользователем при обработке конкретных типов задач или данных. Варианты реализации включают в себя следующее.

* Локальное хранение (LocalStorage, IndexedDB) – используется на клиентской стороне и позволяет сохранять историю параметров непосредственно в браузере, без необходимости взаимодействия с сервером. Подходит для начального этапа и одиночных пользовательских сессий.
* Серверное хранение (PostgreSQL, MongoDB) – применяется в случае централизованного сервиса. Позволяет организовывать долговременное и масштабируемое хранение истории конфигураций, связывая их с пользователем, типом задачи или метаданными.
* Форматы данных – история может сохраняться в виде JSON-объектов, содержащих параметры обработки, их диапазоны, дату и контекст (например, название загруженного скрипта или типа сигнала).

На основе накопленной истории возможно внедрение элементов адаптивного поведения интерфейса. Наиболее простые реализации включают автоматическое восстановление последних использованных параметров при повторной загрузке того же алгоритма, подсказки или автозаполнение диапазонов и значений, ранжирование параметров по частоте использования.

Адаптация поведения системы на основе истории взаимодействия пользователя представляет собой мощный механизм повышения эффективности. Реализация такого подхода позволяет системе перейти от статического инструмента к персонализированному помощнику, способному не просто выполнять команды, а активно предлагать наиболее вероятные успешные конфигурации обработки. Это особенно важно в условиях, когда критерии качества результата не формализуемы, и основную нагрузку по интерпретации несёт пользователь.

В следующей главе рассмотрены технологии, применённые для реализации прототипа: от средств построения интерфейса до исполнения пользовательского кода и локального хранения данных.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Выбор инструментов разработки

### HTML

Для формирования структуры веб-интерфейса использовался язык разметки HTML5 [8]. Он обеспечивает базовую организацию всех визуальных и интерактивных компонентов страницы: размещение графиков, элементов управления параметрами (ползунки, кнопки, выпадающие списки), а также логическое разделение интерфейса на функциональные блоки.

HTML выступает в качестве каркаса приложения, на основе которого реализуются как внешний вид, так и логика поведения элементов. Его универсальность и поддержка всеми современными браузерами делают его естественным выбором при разработке интерфейсов для визуальной настройки алгоритмов.

### SCSS (Sassy CSS)

Для стилизации интерфейса использовался препроцессор SCSS [20], являющийся расширением стандартного CSS. Применение препроцессора позволяет повысить гибкость и модульность оформления, что особенно важно при разработке масштабируемых интерфейсов с повторяющимися стилевыми шаблонами.

В контексте данной системы SCSS обеспечил поддержку единого стилевого оформления для всех компонентов интерфейса и позволил централизованно управлять визуальными изменениями при доработках прототипа.

### JavaScript

Функциональность пользовательского интерфейса реализована с использованием JavaScript [9]. Язык применялся для обработки событий интерфейса (изменение параметров, запуск скриптов), взаимодействия с вычислительным модулем (Pyodide), обновления графиков (Chart.js), а также сохранения состояния в LocalStorage.

Выбор JavaScript обусловлен его универсальностью, прямой интеграцией с HTML/CSS и активной поддержкой со стороны браузера.

### Chart.js

Библиотека Chart.js [3] использовалась для визуализации сигналов и отображения результатов обработки. Она поддерживает построение линейных графиков, обновляемых в реальном времени, масштабирование, аннотирование и интерактивность.

Chart.js легко интегрируется с JavaScript-кодом, позволяет эффективно отрисовывать данные и обеспечивает визуальную основу для экспертной оценки результата пользователем.

### Pyodide

Для исполнения пользовательских алгоритмов, написанных на языке Python, применялся Pyodide [17] – интерпретатор Python, собранный в WebAssembly и работающий в браузере. Это решение позволило временно отказаться от серверной архитектуры и обеспечить:

* исполнение пользовательских скриптов в безопасной среде;
* поддержку библиотек;
* немедленный отклик при изменении параметров;
* гибкость при работе с произвольной логикой обработки данных.

## Техническое задание

### Введение

Настоящее техническое задание (ТЗ) разработано в рамках выполнения выпускной квалификационной работы по направлению 09.04.01 «Информатика и вычислительная техника» и предназначено для формализации требований к программному обеспечению – системе ассистированного преобразования сигналов для задач с плохо формализуемыми метриками качества.

Система, описываемая в данном документе, представляет собой интерактивное программное средство, функционирующее в виде клиентского веб-приложения. Она предназначена для визуальной настройки параметров пользовательских алгоритмов обработки сигналов в условиях, когда результат решения не может быть количественно оценён с помощью стандартных метрик или функций потерь.

В отличие от классических автоматизированных решений, основанных на оптимизации формальных критериев, данная система опирается на экспертную оценку результата и предоставляет пользователю инструменты для пошагового подбора параметров обработки с немедленной визуальной обратной связью. Основное назначение системы заключается не в полной автоматизации процесса обработки, а в ассистировании пользователю, что позволяет ускорить достижение приемлемого с его точки зрения результата.

Система реализуется как MVP (minimum viable product) и ориентирована на работу с одномерными сигналами. Она должна обеспечивать возможность загрузки пользовательского скрипта обработки, визуального управления параметрами этого скрипта и немедленного отображения результата. Особое внимание в проекте уделяется простоте интерфейса, реактивности взаимодействия и возможности накопления пользовательского опыта в форме конфигураций параметров, применённых пользователем при работе с разными сигналами.

В перспективе предполагается расширение функциональности, включая поддержку двумерных и трёхмерных данных, а также обучение персонализированных моделей (нейронных сетей) на основе накопленного опыта, однако данные возможности не входят в рамки текущей реализации и ограничиваются формированием обучающего датасета.

### Основания для разработки

Основанием для разработки является выполнение выпускной квалификационной работы по направлению 09.04.01 «Информатика и вычислительная техника» – «Технологии проектирования системного и прикладного программного обеспечения».

### Назначение разработки

Разрабатываемое программное обеспечение предназначено для поддержки пользовательских сценариев обработки сигналов, в которых результат не может быть однозначно определён формальными критериями. Основная функция системы – обеспечение удобной визуальной среды для настройки параметров пользовательского алгоритма и мгновенного наблюдения за влиянием этих параметров на выходной сигнал.

Программа ориентирована на индивидуальное применение специалистами, работающими с нестандартизированными данными, где анализ результата требует профессионального суждения, а не автоматической оптимизации. Интерфейс и вычислительная модель системы сфокусированы на гибком управлении и наглядности, позволяя пользователю управлять обработкой сигнала без программной доработки инструмента.

### Требования к программе

Программная система должна функционировать в виде веб-приложения, доступного через современный браузер без необходимости установки и подключения к серверной части. В рамках настоящей реализации система ограничивается работой с одномерными сигналами и ориентирована на запуск пользовательского Python-кода в клиентской среде.

Система должна обеспечивать следующую функциональность:

* приём входного файла, содержащего одномерный сигнал в формате .csv, с возможностью визуализации данных в виде графика;
* загрузку пользовательского скрипта на языке Python, содержащего функцию обработки сигнала;
* автоматическое извлечение списка настраиваемых параметров из пользовательского кода;
* представление параметров в интерфейсе в виде элементов управления (ползунки, поля ввода);
* выполнение пользовательской функции обработки в изолированной среде (Pyodide) с передачей текущих параметров;
* отображение результата обработки с возможностью сопоставления с исходным сигналом;
* немедленное обновление визуализации при любом изменении параметров, без ручного запуска;
* локальное сохранение значений параметров и их истории (связка входного сигнала и параметров), пригодных для формирования обучающего набора данных.

В рамках минимально жизнеспособного прототипа:

* поддерживаются только одномерные сигналы;
* обучение на пользовательском опыте не реализуется (производится только сбор данных);
* система не обрабатывает ошибки внутри пользовательского кода автоматически (отладка предполагается со стороны пользователя).

Требования к надёжности и отказоустойчивости:

* система должна корректно отрабатывать некорректные входные данные;
* исполнение пользовательского кода должно происходить в изолированной среде без доступа к файловой системе или системным вызовам;
* все внутренние ошибки (вне пользовательского кода) должны перехватываться и пользователю должно быть выведено сообщение об ошибке.

Программа должна быть работоспособной в актуальных версиях следующих браузеров:

* Google Chrome [26];
* Mozilla Firefox [27];
* Yandex Browser [28].

Мобильная адаптивная версия интерфейса на этапе MVP не требуется.

Таким образом, в рамках этапа проектирования были определены цели разработки, обоснован выбор технологического стека, а также сформулированы функциональные и технические требования к системе. Спроектированная архитектура ориентирована на поддержку пользовательского сценария обработки данных с возможностью загрузки и исполнения произвольных алгоритмов.

В следующей главе подробно рассмотрен процесс реализации программного обеспечения, включая архитектуру клиентской части, механизм исполнения пользовательского кода, организацию взаимодействия модулей, а также особенности визуализации и пользовательского интерфейса.

# РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Структура веб-приложения

### Назначение и принципы построения

Разрабатываемое веб-приложение представляет собой универсальную платформу, ориентированную на запуск и визуальное сопровождение пользовательских вычислительных алгоритмов. В отличие от специализированных решений, система не фиксирована на каком-либо одном классе задач (например, обработке сигналов), а предоставляет инструментарий для воспроизведения и анализа любых корректно оформленных сценариев, реализованных на языке Python. Пользователь, обладающий собственным набором данных и функцией обработки, может получить интерактивный результат без необходимости настройки среды разработки или установки зависимостей.

Ключевой особенностью проекта является его самодостаточность: все действия (от загрузки данных и скрипта до отображения результата) происходят на стороне клиента. Это достигается за счёт использования технологии WebAssembly и интерпретатора Pyodide, позволяющего выполнять Python-код прямо в браузере.

С точки зрения архитектуры, приложение построено по модели одностраничного интерфейса (SPA, Single Page Application), в котором логика взаимодействия разнесена по последовательным этапам. Каждый этап представлен в DOM как отдельная секция, визуально активизируемая при переходе пользователя. Таким образом, пользовательский путь строго упорядочен, а интерфейс остаётся компактным и целостным.

При разработке соблюдаются следующие принципы.

* Локальность вычислений. Все данные и скрипты остаются в браузере пользователя, что исключает риски утечки и снижает технические требования к серверной инфраструктуре.
* Модульность и масштабируемость. Архитектура позволяет при необходимости расширить проект за счёт новых этапов, параметров или типов визуализации без переписывания существующих модулей.
* Простота подключения пользовательского кода. Загрузка и исполнение скрипта не требуют от пользователя знаний о внутреннем устройстве системы – достаточно следовать простому формату функции.
* Универсальность и независимость от предметной области. Приложение не жёстко привязано к обработке сигналов, временных рядов или иных типов данных, что делает его применимым в широком спектре задач: от статистического анализа до предобработки обучающих выборок.

Таким образом, назначение системы заключается в том, чтобы предоставить единое визуально-ориентированное средство запуска пользовательских Python-функций, способное адаптироваться под произвольный сценарий и обеспечить обратную связь в удобной для анализа форме.

### Компоненты и структура проекта

Архитектура проекта подразумевает чёткое разделение логики по функциональным блокам, каждый из которых отвечает за конкретный аспект работы системы. Это обеспечивает прозрачность реализации, упрощает сопровождение кода и позволяет легко расширять функциональность. Все компоненты работают в рамках одностраничного веб-приложения и взаимодействуют друг с другом без участия серверной стороны.

Основу интерфейса составляет файл index.html, содержащий структуру всех этапов пользовательского взаимодействия. Каждый шаг реализован в виде <section> с уникальным идентификатором (step1, step2, step3, step4) и включает в себя элементы управления, формы, визуальные блоки и вспомогательные тексты. Активация нужного шага происходит путём изменения классов видимости на уровне DOM.

Стилизация интерфейса реализована с использованием SCSS-модулей, сгруппированных по зонам ответственности: общая стилистика (\_main.scss), боковая панель (\_sidebar.scss), кнопки (\_button.scss), визуальные секции (\_section.scss). Это позволяет поддерживать единый визуальный стиль при сохранении гибкости и возможности адаптации под разные разрешения экрана.

Вся логика взаимодействия реализована на клиентской стороне с помощью JavaScript и разделена на отдельные модули:

* sidebar.js – основной управляющий модуль, отвечающий за переходы между шагами, загрузку файлов и формирование вызова Python-функции;
* chart.js – модуль визуализации, использующий библиотеку Chart.js для отрисовки данных;
* pyRunner.js – модуль взаимодействия с интерпретатором Pyodide: инициализация, передача кода, обработка результата.

Такой подход позволяет поддерживать масштабируемую структуру, в которой каждый модуль можно переиспользовать или заменить без влияния на остальные части проекта.

Для хранения текущего состояния (выбранная функция, загруженные данные, параметры) используется глобальный объект window, а DOM-элементы синхронизируются с этим состоянием через события и обновление значений форм. Это упрощает обработку взаимодействий без привлечения внешнего фреймворка и делает поведение приложения предсказуемым.

### Поток данных и взаимодействие модулей

Взаимодействие между частями приложения организовано по принципу последовательного прохождения данных сквозь этапы пользовательского сценария. При этом каждый модуль отвечает не только за выполнение своей задачи, но и за подготовку выходных данных для следующего шага. Такое построение соответствует принципу однонаправленного потока данных, характерному для реактивных клиентских приложений.

Этапы передачи данных включают в себя следующее.

1. Загрузку входного файла. Пользователь загружает файл формата CSV через интерфейс первого шага. Объект File, полученный из DOM, сохраняется в глобальном пространстве браузера (переменная window.uploadedFile) и далее используется без повторной загрузки.
2. Разбор пользовательского скрипта. На втором шаге пользователь загружает .py-файл, который читается как текст и сохраняется в переменную window.pyContent. Анализ кода производится с помощью регулярного выражения, извлекающего имена функций и их аргументы.
3. Формирование вызова функции. На третьем шаге пользователь выбирает одну из найденных функций и задаёт значения параметров (в том числе диапазоны). Эти параметры сериализуются в массив window.currentArguments, а имя функции сохраняется в window.currentFunction.
4. Передачу в Pyodide. После перехода к четвёртому шагу входной файл записывается во внутреннюю файловую систему Pyodide (виртуальный FS), скрипт исполняется, и вызывается функция с переданными параметрами. Результат (таблица в формате pandas.DataFrame) конвертируется в CSV и возвращается обратно в JavaScript.
5. Обработку и визуализацию результата. Возвращённый CSV разбирается с помощью JS и используется для построения графика. Если параметр предполагает интерактивное управление, то при его изменении функция вызывается повторно с новым значением, что позволяет реализовать реактивную модель визуализации.

Для синхронизации данных между модулями используется явное хранение состояния в глобальных переменных, что обеспечивает доступ к ним из любых частей проекта. В случае возникновения ошибок на любом этапе (некорректный формат файла, ошибка исполнения скрипта, отсутствие возвращаемого значения), срабатывает обработчик исключений, выводящий поясняющее сообщение и не блокирующий дальнейшие действия пользователя.

## Поддержка пользовательских данных и сценариев

### Типы поддерживаемых данных

Система ориентирована на приём и обработку файлов в текстовом формате CSV (Comma-Separated Values), что обеспечивает совместимость с большинством программ для работы с табличными данными (например, Microsoft Excel, Google Sheets, Jupyter Notebook). Использование CSV обусловлено его простотой, читаемостью и универсальностью.

Файл должен удовлетворять следующим минимальным требованиям:

* Заголовок столбцов. Первая строка файла должна содержать названия всех полей. Это необходимо для последующей интерпретации и визуализации данных, особенно при возврате результата в виде таблицы.
* Хотя бы один числовой столбец. Алгоритмы обработки и визуализации требуют наличия численных значений, поэтому хотя бы один столбец должен содержать числа с плавающей точкой или целые значения.
* Допустимость пропусков. Система корректно обрабатывает NaN, пустые ячейки или строки, содержащие пропуски. Однако их интерпретация и восстановление должны быть реализованы в пользовательском скрипте.
* Одинаковая длина строк. Все строки (включая строки с пропусками) должны содержать одинаковое количество разделённых значений, чтобы исключить синтаксические ошибки при разборе файла.
* Кодировка UTF-8. Для корректной обработки в браузере и интерпретаторе Pyodide используется UTF-8. Файлы с иной кодировкой могут привести к ошибкам чтения.

Система не накладывает ограничений на название полей, количество столбцов или смысл данных. Однако визуализация графика на текущий момент реализована только для одного числового ряда, поэтому предпочтительно размещать в файле два столбца: идентификатор (например, индекс или время) и значение.

Дополнительные поля не препятствуют работе, но могут быть проигнорированы на этапе построения графика, если не используются в логике пользовательского скрипта.

### Обработка и валидация данных на клиенте

Поскольку система работает полностью на стороне клиента, все проверки и предварительная обработка входных данных происходят непосредственно в браузере пользователя, без обращения к серверу. Это требует от интерфейса устойчивости к ошибкам, способности выявлять потенциальные проблемы до запуска алгоритма и предоставлять понятные сообщения в случае некорректного ввода.

После загрузки CSV-файла система вначале проверяет его расширение. Если формат не соответствует ожидаемому, пользователь получает уведомление об ошибке. Далее файл считывается в память при помощи объекта FileReader, и его содержимое интерпретируется как обычный текст. Выполняется поверхностный анализ: проверяется наличие заголовков, допустимость пропусков, соответствие количества столбцов в каждой строке и корректность синтаксиса разделителей.

Если структура файла вызывает сомнения, то система информирует пользователя и не позволяет продолжить работу. При успешной загрузке отображается имя файла, становится доступной кнопка перехода к следующему шагу, а также возможность предварительного просмотра графика.

Важно, что на этом этапе система не изменяет и не интерпретирует данные. Пропущенные значения, нечисловые поля и другие особенности остаются в исходном виде. Это решение было принято для того, чтобы не навязывать пользователю модель обработки и оставить свободу реализации собственных алгоритмов в Python-скрипте.

Ошибки чтения, синтаксические сбои или попытка загрузить файл с неподдерживаемой структурой сопровождаются поясняющими сообщениями, при этом дальнейшее использование интерфейса остаётся доступным. Такой механизм защищает систему от сбоев, не перегружает пользователя техническими деталями и сохраняет гибкость архитектуры.

### Универсальность ввода

Одной из важнейших характеристик разрабатываемого приложения является независимость от конкретной предметной области. В отличие от специализированных систем, разработанных для узкого круга задач (например, обработки сигналов, графиков, спектров и т. д.), данное решение ориентировано на работу с любыми пользовательскими алгоритмами, оформленными в виде Python-функций.

Система не предполагает заранее известной структуры входных данных или фиксированного набора параметров. Вместо этого она предоставляет механизмы передачи содержимого CSV-файла и аргументов, введённых пользователем, напрямую в функцию, определённую в загруженном скрипте. Такой подход позволяет обрабатывать не только числовые ряды, но и другие формы представления информации, если пользовательская функция реализует необходимую логику.

Универсальность достигается также за счёт возможности указания диапазонов значений параметров и динамического создания элементов управления (например, ползунков), что делает интерфейс адаптивным к любому количеству и типу аргументов.

Таким образом, система представляет собой платформу, способную масштабироваться под произвольную задачу без модификации исходного кода, что особенно важно в контексте обработки некорректных или нестандартизированных данных.

## Реализация поддержки Python-алгоритмов

### Обработка пользовательских скриптов и извлечение функций

Поддержка произвольных пользовательских алгоритмов реализована через механизм загрузки .py-файла, содержащего одну или несколько функций, оформленных по стандартам языка Python. Интерфейс системы не требует выполнения кода на момент загрузки – вместо этого производится анализ текста скрипта, направленный на извлечение доступных для вызова функций и их аргументов.

После загрузки файл читается как строка и сохраняется в оперативной памяти браузера. На этом этапе производится синтаксический разбор с помощью регулярного выражения, ориентированного на конструкции вида def имя\_функции(аргументы). Извлекаются имена всех функций верхнего уровня, за исключением тех, которые содержат необязательные параметры, ссылки на self или аннотации, несовместимые с целью вызова.

В результате анализа формируется список доступных функций и список их аргументов. Эти данные используются для генерации интерфейса на следующем этапе, где пользователь выбирает одну из доступных функций и указывает значения параметров.

Важно отметить, что разбор кода на этом этапе осуществляется вне интерпретатора Pyodide, что позволяет выявлять потенциальные ошибки структуры до выполнения скрипта. Такой подход защищает систему от сбоев и гарантирует, что только корректно оформленные функции попадут в дальнейшую обработку.

Поскольку проект ориентирован на универсальность, не накладываются ограничения на название функций или состав аргументов. Однако предполагается, что одна из функций возвращает результат в виде объекта pandas.DataFrame или эквивалентной структуры, пригодной для сериализации в CSV и последующей визуализации.

### Универсальная модель вызова функции

Одной из задач при разработке интерфейса стало построение такой логики, которая позволила бы вызывать любую пользовательскую функцию без предварительного знания её имени, количества аргументов и порядка их передачи. Для этого была реализована универсальная модель формирования вызова, основанная на данных, полученных при разборе .py-файла.

После выбора пользователем одной из загруженных функций, система автоматически формирует строку вызова в формате Python. Первый аргумент, как правило, представляет собой загруженный CSV-файл, который передаётся в виде объекта open('input.csv', 'rb'). Остальные параметры (числовые значения или диапазоны), заданные через форму, сериализуются в соответствующие аргументы. Например, при наличии параметра order, система передаёт его как целое число, считанное с ползунка.

Важным моментом является то, что система не требует от функции строго фиксированного набора параметров. Если функция принимает только один аргумент (входной файл), то она будет вызвана без дополнительных параметров. Если в сигнатуре присутствуют числовые значения, они передаются согласно их позициям, что делает возможным использование скриптов разной сложности, от простых фильтров до моделей с несколькими настраиваемыми коэффициентами.

Таким образом, построенная модель вызова позволяет абстрагироваться от содержимого конкретной функции и обрабатывать произвольный пользовательский код в единообразной форме, при этом поддерживая передачу динамических параметров и совместимость с различными форматами обработки.

### Поддержка аргументов и интерактивных примеров

После выбора пользователем одной из функций, загруженных из скрипта, система формирует соответствующую форму ввода параметров. Поля создаются динамически на основе анализа аргументов, обнаруженных в сигнатуре функции. Первый из них, как правило, представляет собой файл с данными и подставляется автоматически, тогда как остальные отображаются в виде интерактивных элементов, доступных для редактирования.

Для числовых параметров предусмотрен ввод диапазона (минимального и максимального значения). Это даёт возможность в дальнейшем сгенерировать на шаге визуализации управляющий элемент, такой как ползунок. Его перемещение изменяет значение параметра и вызывает функцию повторно, что позволяет быстро наблюдать, как чувствителен результат к изменениям заданных настроек.

Аргументы, не затронутые ползунком, передаются в функцию один раз и остаются постоянными в течение всей сессии. Это упрощает реализацию как со стороны интерфейса, так и внутри пользовательского алгоритма. В результате достигается баланс между универсальностью и удобством настройки. Параметры можно задавать один раз, а можно исследовать динамически.

### Изоляция среды исполнения и безопасность

Запуск пользовательских алгоритмов осуществляется в среде Pyodide, которая представляет собой интерпретатор Python, собранный в WebAssembly и выполняющийся внутри браузера. Благодаря этому подходу выполнение кода полностью изолировано от операционной системы пользователя и не требует обращения к серверу, что исключает большинство рисков, связанных с безопасностью.

Среда Pyodide не предоставляет доступ к файловой системе, сетевым интерфейсам, командной строке или системным библиотекам. Такие модули, как os, sys, subprocess либо отключены полностью, либо доступны в урезанном варианте, не позволяющем влиять на окружение за пределами интерпретатора. Это означает, что даже если пользователь загрузит потенциально вредоносный скрипт, он не сможет выполнить действия за пределами виртуальной среды.

Кроме того, система блокирует загрузку дополнительных пакетов, кроме тех, что явно разрешены. Например, попытка импортировать стороннюю библиотеку без её предварительной загрузки через micropip приведёт к ошибке выполнения. По умолчанию доступны только базовые модули: numpy, pandas, matplotlib, scipy и несколько вспомогательных библиотек. Такой набор достаточен для большинства аналитических задач, но при этом не предоставляет инструментов, которые могли бы быть использованы для обхода ограничений браузера.

Сценарий выполнения устроен таким образом, что каждый вызов функции создаёт новую интерпретацию данных внутри изолированной сессии. Код исполняется последовательно, а результат возвращается только в текстовом виде, что полностью исключает возможность утечки информации или перехвата управления. Даже при возникновении ошибок (например, ImportError, ValueError или SyntaxError) поведение приложения предсказуемо: пользователь получает уведомление, а выполнение функции прекращается.

Изоляция среды позволяет безопасно обрабатывать загруженные скрипты без необходимости предварительной модерации или анализа кода, что особенно важно для системы, предназначенной для широкой аудитории с произвольными задачами.

## Обработка результата и визуальное представление

### Форматы выходных данных и поддерживаемые структуры

После выполнения пользовательской функции в среде Pyodide результат возвращается в систему в виде сериализованного текстового представления. Для обеспечения совместимости и надёжности обработки, система поддерживает лишь один формат возвращаемого значения – табличную структуру pandas.DataFrame.

Ограничение связано с тем, что DataFrame позволяет формализовать результат в универсальной форме: таблицы с именованными столбцами и строками, где каждый столбец может быть интерпретирован как серия данных для визуализации или дальнейшего анализа. Кроме того, pandas предоставляет встроенный метод to\_csv(), позволяющий легко и однозначно преобразовать таблицу в текст, который затем считывается средствами JavaScript на стороне клиента.

Для успешной визуализации результат должен удовлетворять следующим требованиям:

* содержать не менее одного числового столбца (float или int);
* не быть пустым (отсутствие строк или только NaN-значения приведёт к сбою);
* иметь читаемые заголовки столбцов (для использования в графике);
* быть корректно сериализуемым методом to\_csv() без параметров index=True.

Скрипты, возвращающие другие типы данных (например, списки, словари, строки или None), не могут быть интерпретированы системой. В таких случаях будет выведено сообщение об ошибке, а выполнение останется безопасным и изолированным.

Таким образом, единый формат возврата результата упрощает как архитектуру интерфейса, так и опыт пользователя: вне зависимости от логики внутри скрипта, если он возвращает DataFrame, система сможет его отобразить. Это создаёт стандарт взаимодействия между пользовательским кодом и системой без необходимости ручного формата данных.

### Механизм передачи результата из Python в JavaScript

Поскольку система работает целиком в браузере, передача данных между средой выполнения Python (Pyodide) и интерфейсом, написанным на JavaScript, должна быть максимально прямолинейной и надёжной. Для этого используется метод to\_csv() из библиотеки pandas, который преобразует результат выполнения функции в строку с разделителями (стандартный формат CSV).

Таким образом формируется строка, содержащая все значения таблицы, включая заголовки столбцов, но без лишнего индекса. Эта строка передаётся обратно в JavaScript через метод runPythonAsync(), возвращающий результат исполнения Python-кода как обычную строку.

На стороне JavaScript эта строка воспринимается как сериализованный CSV-файл. Она разбивается по строкам, затем каждая строка по запятым, и формируется структура, пригодная для дальнейшей обработки: построения графика, отображения в таблице или сохранения.

Передача реализована в текстовом виде без промежуточного биндинга типов, что исключает проблемы совместимости. Используются не бинарные форматы, JSON или объекты со сложной структурой, а исключительно строки, пригодные для разбора стандартными методами.

Такой механизм обеспечивает надёжную, прозрачную и платформонезависимую связь между двумя языковыми средами и делает возможным обработку произвольных данных без необходимости внедрения сторонних решений для сериализации.

### Визуализация и масштабируемость

Графическая визуализация результатов анализа реализована с использованием библиотеки Chart.js – одного из наиболее популярных решений для построения интерактивных графиков в браузере. Она обеспечивает гибкий и в то же время производительный механизм отрисовки данных, достаточный для большинства аналитических задач.

После передачи результата из среды Python, система разбирает CSV-строку и извлекает два ключевых массива: массив подписей по оси X (например, индексы или временные метки) и массив числовых значений по оси Y. На основе этих данных создаётся график типа line, который размещается в заранее определённом контейнере на странице.

Chart.js автоматически масштабирует график по размеру контейнера, поддерживает анимации, сглаживание линий (tension) и изменение диапазонов осей. Это делает отображение наглядным даже при больших объёмах данных (до нескольких тысяч точек). При необходимости могут быть добавлены фильтрация, агрегация или оконное сглаживание, но базовая реализация работает без потери производительности для типовых объёмов.

Визуализация также интегрирована с механизмом управления параметрами: если пользователь изменяет значение параметра через ползунок, система пересчитывает результат и заново отрисовывает график. При этом старое изображение уничтожается, и создаётся новая диаграмма на основе обновлённого набора данных. Такой подход обеспечивает высокую интерактивность и позволяет исследовать влияние параметров на результат анализа в реальном времени.

Поскольку Chart.js не зависит от серверной части и работает только с JavaScript-объектами, его использование полностью совместимо с моделью одностраничного приложения и изолированной среды исполнения.

Полный код программы приведён в Приложении 1.

## Отчёт о реализации и оценка функциональности

### Поддерживаемые сценарии использования

Разработанная система ориентирована на широкий спектр пользовательских задач, которые можно выразить в виде функции, принимающей табличные данные и возвращающей результат в табличной форме. Несмотря на универсальный характер платформы, в процессе разработки и тестирования были реализованы конкретные сценарии, подтверждающие её функциональность.

Один из базовых примеров – восстановление пропущенных значений в числовом ряду. Пользователь может загрузить CSV-файл с неполными данными, выбрать подходящий метод интерполяции (например, линейный или полиномиальный) и указать степень аппроксимации. Получившийся результат немедленно отображается на графике, позволяя сравнивать разные подходы визуально.

Другим сценарием служит анализ тренда или флуктуаций во временных рядах. Пользовательский скрипт может включать расчёт скользящего среднего, дисперсии или других статистических показателей. Такие операции не требуют изменения интерфейса, поскольку система абстрагирована от назначения функции: если скрипт возвращает корректную таблицу, результат будет отображён.

Дополнительно реализована возможность параметризации вычислений. Пользователь может задать диапазон значений одного из параметров и управлять им с помощью ползунка. Это позволяет использовать систему для обучения, сравнения методов или поиска оптимальных параметров модели.

Благодаря поддержке произвольного кода система применима и к более сложным задачам: агрегации, фильтрации, кластеризации, генерации признаков, преобразованию формата данных. Единственным требованием остаётся соблюдение стандарта входа (CSV) и выхода (DataFrame).

### Ограничения текущей версии

Несмотря на универсальность архитектурного решения, текущая версия системы обладает рядом ограничений, связанных как с техническими особенностями, так и с выбранными допущениями при реализации прототипа.

Во-первых, поддерживается только одномерный формат данных – система предполагает, что загружаемый CSV-файл содержит одну числовую колонку. Это упрощает визуализацию, но ограничивает применение многомерных функций или анализа зависимости между переменными. Расширение функциональности под многоколонковые таблицы требует переработки интерфейса построения графиков и формы задания параметров.

Во-вторых, все действия выполняются вручную. Пользователь самостоятельно загружает данные и скрипты, задаёт параметры, инициирует переход между шагами. Нет поддержки автоматической загрузки из внешних источников, drag-and-drop интерфейса или сохранения предыдущих сессий.

Также отсутствует механизм сохранения и кэширования промежуточных результатов. Повторный запуск одного и того же скрипта с теми же параметрами требует повторного исполнения, что может быть неэффективно при тяжёлых вычислениях. Аналогично, пользователь не может вернуться к ранее построенному графику без полного пересчёта.

Интерфейс системы не предусматривает редактирования или предпросмотра кода, а также не поддерживает подсветку синтаксиса, отладку или вывод промежуточных значений. Это делает использование системы удобным для готовых и отлаженных скриптов, но не оптимальным для разработки с нуля.

Наконец, текущая реализация ориентирована на одиночный результат: функция должна возвращать единственный DataFrame, пригодный для преобразования в CSV и отображения. Модели, возвращающие сложные структуры, графы, изображения или интерактивные элементы, не поддерживаются.

### Направления развития

В процессе эксплуатации и анализа архитектуры системы были выделены ключевые направления развития, которые позволят превратить прототип в более гибкий, надёжный и масштабируемый инструмент.

Прежде всего, целесообразно внедрение поддержки многошаговых вычислений. Добавление цепочек вызовов или возможность этапной обработки (например, фильтрация, интерполяция, агрегация) расширит сферу применения и позволит формировать полноценные сценарии анализа.

Следующим шагом может стать встроенный редактор кода с базовыми возможностями редактирования .py-файлов прямо в интерфейсе. Это упростит проверку и отладку скриптов, особенно для пользователей, не имеющих внешней среды разработки. Подсветка синтаксиса, отображение ошибок в реальном времени и шаблоны функций повысят удобство работы.

Также рассматривается возможность автоматического определения структуры функции и генерации параметров по аннотациям. Если пользователь укажет типы аргументов с помощью Python type hints, система сможет создавать более точные поля ввода, включая строки, булевы значения, выбор из списка и другие типы.

Для упрощения настройки предлагается реализовать библиотеку шаблонов параметров, где пользователь сможет выбирать предустановленные конфигурации или сохранять свои собственные. Это особенно актуально для часто повторяющихся сценариев или при работе в учебной среде.

Потенциальным направлением является расширение визуализации: добавление поддержки других типов графиков (гистограммы, области, тепловые карты), управления осями и легендами, а также экспорт результатов в виде изображений или таблиц.

Одним из перспективных направлений развития является переход к клиент-серверной архитектуре. В текущей реализации всё вычисление и визуализация происходят исключительно на стороне клиента, что ограничивает доступ к длительным или ресурсоёмким вычислениям. Внедрение серверной части позволило бы реализовать кэширование результатов, параллельную обработку, а также подключение специализированных вычислительных библиотек, не совместимых с Pyodide. Кроме того, это создаёт возможности для управления сессиями, загрузки истории, коллективной работы и интеграции с базами данных.

Не менее важным направлением является накопление пользовательского опыта. При сохранении истории параметров, используемых в скриптах, и соответствующих визуальных результатов, можно формировать выборку, пригодную для последующего анализа и обучения. Это, в свою очередь, открывает возможность внедрения нейросетевого помощника, способного рекомендовать значения параметров на основе предыдущих успешных запусков [30].

Модель может обучаться на данных о зависимости параметров от типа данных и характера результата (например, плавность кривой, отсутствие артефактов). В перспективе это позволит реализовать режим интеллектуальной подсказки: система будет предлагать параметры, которые с высокой вероятностью обеспечат качественный результат без необходимости ручного перебора.

Таким образом, эволюция системы может происходить как в направлении повышения вычислительной мощности и устойчивости (за счёт серверной части), так и в сторону интеллектуализации пользовательского взаимодействия на основе накопленных знаний и машинного обучения. Эти улучшения могут быть реализованы поэтапно, без нарушения текущей архитектуры, что делает систему пригодной для дальнейшего развития без кардинальной переработки.

## Демонстрационный пример

Для иллюстрации принципов работы системы был подготовлен демонстрационный сценарий, в котором пользователь восстанавливает недостающие значения в числовом ряду с помощью загруженного Python-алгоритма. В примере поочерёдно используются все функциональные этапы интерфейса, каждый из которых сопровождается визуальной обратной связью.

На Рис. 3.1 продемонстрирована главная страница системы, содержащая первый этап работы («Загрузка данных»).

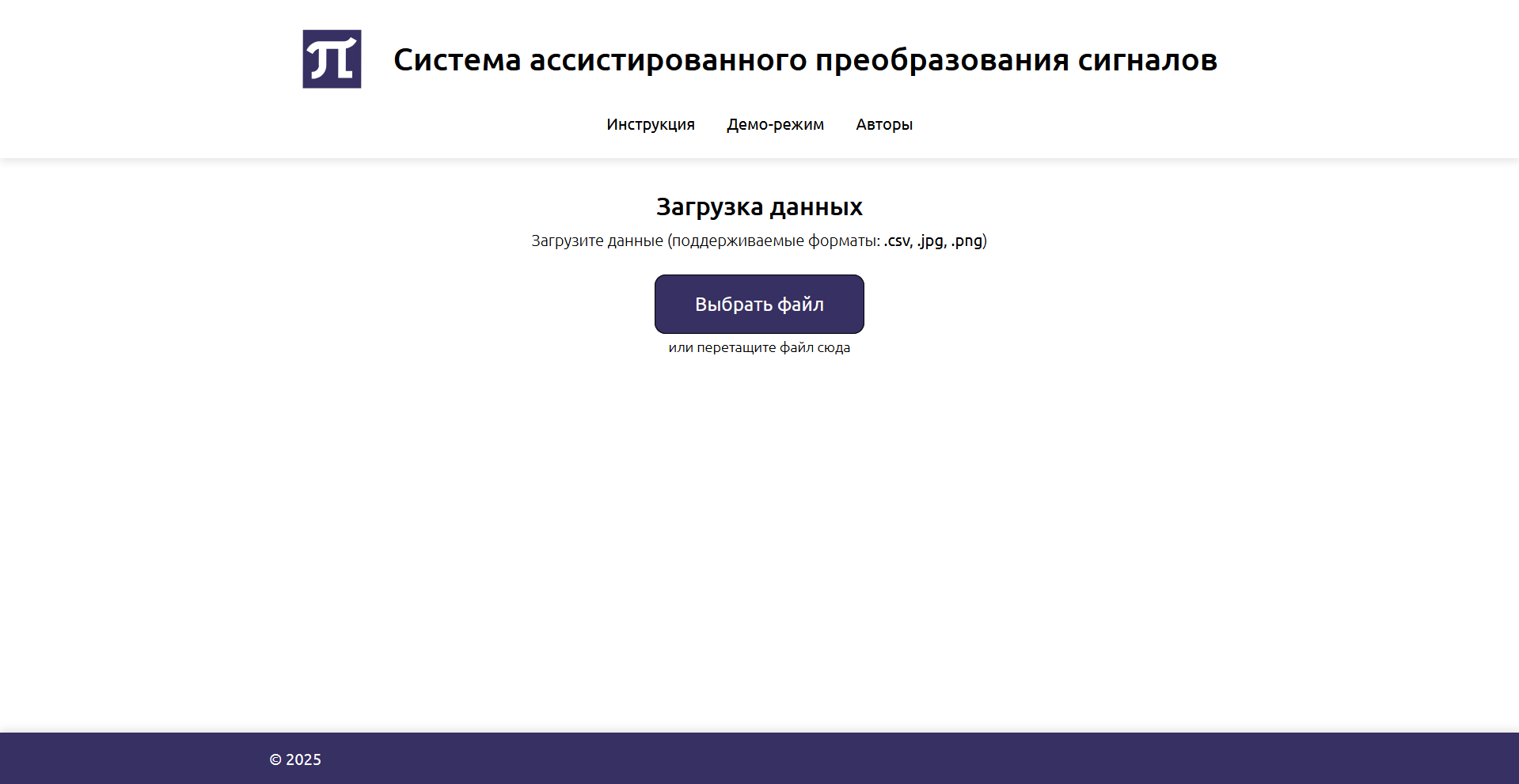


Рис. . – Этап «Загрузка данных»

Первым шагом является загрузка файла в формате CSV, содержащего одномерный числовой ряд с пропущенными значениями (Рис. 3.2). Пример содержимого CSV-файла представлен на Рис. 3.3. Интерфейс отображает название файла и предлагает визуализировать данные, по которым сразу можно определить структуру сигнала и участки пропусков (Рис. 3.4).

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Загрузка CSV-файла на этапе «Загрузка данных»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Пример структуры CSV-файла

Изображение выглядит как текст, снимок экрана, График, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Визуализация входных данных

На втором этапе загружается скрипт, написанный пользователем (файл с расширением .py, содержащий несколько функций обработки). Этот процесс демонстрируется на Рис. 3.5.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Загрузка скрипта на этапе «Загрузка скрипта обработки данных»

Третий этап – настройка параметров. Пользователю предлагается форма с радиокнопками для выбора нужной функции и полями ввода для задания диапазона параметров (Рис. 3.6). При указании min и max значений параметра order интерфейс формирует ползунок, позволяющий плавно регулировать значение в заданных пределах.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Выбор функции и параметров

На финальном, четвёртом этапе, выполняется вызов выбранной функции с заданными параметрами. Результат отображается в виде графика, на который автоматически накладываются восстановленные данные (Рис. 3.7). Пользователь может изменить значение параметра order, перемещая ползунок. Данное действие немедленно приводит к повторному вызову функции и обновлению графика.

Изображение выглядит как текст, снимок экрана, График, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Страница итогового шага «Результат»

Пример применения системы на задаче обратной фильтрации изображения приведён в Приложении 2. UML-диаграмма активности системы приведена в Приложении 3.

# ТЕСТИРОВАНИЕ И АПРОБАЦИЯ

## Статический анализ

### Инструмент статического анализа

В рамках проекта особое внимание уделялось качеству клиентской части веб-приложения, реализованной на языке JavaScript. Для обеспечения надёжности и соответствия кода современным стандартам был использован инструмент статического анализа ESLint [6]– одно из наиболее популярных решений в экосистеме JavaScript.

ESLint позволяет выполнять автоматическую проверку исходного кода без его выполнения, выявляя синтаксические ошибки, потенциально уязвимые конструкции, нарушения стиля, неиспользуемые переменные и дублирование логики. Инструмент предоставляет гибкую систему настройки правил, поддержку популярных стилевых соглашений (например, Airbnb, Google, StandardJS) и легко интегрируется в редакторы кода, включая Visual Studio Code.

Выбор ESLint обусловлен следующими причинами:

* широкая поддержка в сообществе и активное развитие;
* возможность адаптации к специфике проекта за счёт подключения или отключения отдельных правил;
* интеграция с редактором VS Code, обеспечивающая немедленную обратную связь при редактировании кода;
* наличие командной строки для анализа всех JavaScript-файлов проекта и формирования отчётов.

Конфигурация ESLint была создана с использованием встроенного мастера и основана на стиле Airbnb. Это позволило унифицировать структуру кода, сократить количество логических ошибок и повысить читаемость отдельных модулей (sidebar.js, chart.js, и др.). В процессе работы ESLint выполнял роль не только средства проверки, но и рекомендационного помощника, предлагающего правки для повышения качества проекта. Конфигурация ESLint представлена на Рис. 4.1.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Конфигурация ESLint

### Результаты проверки и типичные замечания

В рамках статического анализа JavaScript-кода были проанализированы основные модули проекта, включая sidebar.js и chart.js. Для анализа использовался инструмент ESLint с базовой конфигурацией, основанной на рекомендованных правилах (eslint:recommended). В результате первичной проверки были зафиксированы идентичные ошибки в обоих файлах (Рис. 4.2).

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Ошибки в файлах при статическом анализе

Эта ошибка указывает на то, что ESLint обнаружил использование конструкции import или export (ES-модулей), но не был настроен на работу с таким синтаксисом. По умолчанию ESLint ожидает, что код будет использовать CommonJS или глобальные переменные браузера. Если не указать явно, что код написан в формате ECMAScript-модулей, линтер считает import и export синтаксической ошибкой.

В техническом плане проблема заключается в отсутствии настройки parserOptions.sourceType = "module" в конфигурации ESLint или в неправильном распознавании самой конфигурации. Также возможны конфликты между версией ESLint и используемым форматом .eslintrc (например, при использовании module.exports в файле без указания типа commonjs в package.json).

Эта ошибка не связана с самим содержанием кода – конструкции import и export корректны и используются в соответствии с современными стандартами JavaScript.

## Модульное тестирование

### Обоснование выбора инструмента

Для проведения модульного тестирования JavaScript-компонентов системы был выбран инструмент Jest [10] – современная среда тестирования, ориентированная на простоту использования и высокую производительность [10]. Данный фреймворк активно применяется в индустрии, поддерживается сообществом и официально рекомендован для проектов, построенных на чистом JavaScript или с использованием React и других популярных библиотек.

В пользу выбора Jest относят следующие факторы.

* Поддержка модульной структуры. Jest корректно обрабатывает ECMAScript-модули (import/export), что особенно важно в условиях использования современной фронтенд-архитектуры.
* Автоматическое окружение. Jest не требует внешней настройки среды исполнения или привязки к браузеру – все тесты выполняются в изолированном виртуальном окружении.
* Простота конфигурации. Установка и запуск тестов сводятся к минимуму команд, без необходимости дополнительной сборки.
* Встроенные инструменты. Jest включает средства мокирования функций, проверки вызовов и генерации отчётов о покрытии кода.
* Совместимость с Node.js. Полная интеграция с npm-экосистемой позволяет подключать Jest к существующим проектам без модификации основного кода.

Jest автоматически находит все файлы с суффиксом .test.js или .spec.js, и запускает их в тестовой среде. Это позволяет эффективно проверять отдельные модули системы на корректность логики без запуска всего приложения.

### Сценарии тестирования и проверка логики функций

Для подтверждения корректности реализации клиентской логики в системе были проведены модульные тесты JavaScript-компонента sidebar.js, отвечающего за обработку пользовательского ввода и переключение состояний интерфейса. Тестирование выполнялось с использованием библиотеки Jest в сочетании с виртуальной средой jsdom, которая позволяет эмулировать DOM API браузера в среде Node.js.

Были выделены четыре группы сценариев, отражающие ключевые аспекты работы клиентского интерфейса.

1. Базовое взаимодействие пользователя с интерфейсом, включая нажатие на кнопку загрузки файла, вызов системного диалога, обработку неподдерживаемых форматов и проверку появления/скрытия кнопки визуализации в зависимости от типа файла.
2. Загрузка и анализ пользовательского скрипта, в рамках которого проверяется корректность отображения функций из .py-файла, наличие параметров и реакция интерфейса при отсутствии корректной структуры.
3. Переходы между этапами пользовательского сценария: проверяется, что переход от шага к шагу осуществляется при наличии необходимых данных, а также возможен возврат к предыдущим шагам без потери состояния.
4. Работа с параметром, включая отображение ползунка, передачу значения в алгоритм и обновление графика при изменении значения. Дополнительно тестируется визуальная обратная связь в виде числового индикатора рядом со слайдером.

Каждый из тестов проверял наличие ожидаемой реакции на соответствующее действие: вызов метода click у элемента ввода файла, показ предупреждения alert, изменение свойства style.display у кнопки графика.

Перед выполнением каждого теста загружалась минимальная HTML-разметка, имитирующая основную структуру страницы, включая необходимые элементы:

* #fileInput – элемент выбора файла;
* .data\_\_button – кнопка загрузки;
* #plotButton – кнопка построения графика;
* другие вспомогательные DOM-элементы.

Также были использованы заглушки (mock) для браузерных методов, таких как alert, click, dispatchEvent. Это позволило запускать тесты изолированно, без зависимости от внешнего окружения браузера.

Все тесты завершились успешно. Отчёт выполнения, сформированный Jest, представлен на Рис. 4.3.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Отчёт модульного тестирования

Таким образом, реализация интерфейсного модуля показала стабильное и предсказуемое поведение при эмуляции пользовательских действий. Структура кода и его организация позволяют эффективно покрывать поведение отдельных компонентов модульными тестами, что является положительным фактором с точки зрения сопровождения проекта и его последующего масштабирования.

Модульное тестирование подтвердило, что базовые сценарии работы с интерфейсом реализованы корректно. Выделение логики обработки событий и состояния в отдельных элементах DOM позволило формализовать поведение системы и облегчить процесс тестирования. При необходимости тесты могут быть расширены за счёт подключения инструментов анализа покрытия и проверки граничных условий.

## Динамический анализ

### Цели динамического анализа и выбранный инструмент

Динамический анализ кода направлен на выявление потенциальных ошибок, утечек памяти, неэффективных операций и нестабильных состояний во время выполнения приложения. В отличие от статического анализа, он позволяет зафиксировать поведение системы в реальном времени – в том числе в условиях,

Для проведения динамического анализа JavaScript-кода в рамках данного проекта был выбран инструмент Firefox Developer Tools [7], встроенный в браузер Mozilla Firefox. Этот инструмент предоставляет:

* профилирование производительности JavaScript-кода и оценки времени отрисовки интерфейса (вкладка Profiler);
* мониторинг использования памяти, снимки состояния кучи и анализ утечек (вкладка Memory);
* отслеживание количества активных обработчиков событий и DOM-событий (вкладка Event Listeners во вкладке Inspector);
* анализ сетевых запросов и передаваемых данных (вкладка Network);
* отладка выполнения скриптов, просмотр стека вызовов и установка точек останова (вкладка Debugger).

Выбор Firefox Developer Tools обусловлен его точной интеграцией с современными веб-API, расширенной поддержкой работы с Pyodide и WebAssembly, а также наличием полнофункциональных средств анализа поведения одностраничных интерфейсов [12]. Инструмент не требует дополнительной настройки и подходит для анализа производительности, отзывчивости и корректности исполнения пользовательского кода прямо в браузере.

### Сценарии профилирования и результаты

Для оценки производительности системы в реальных условиях был использован профайлер браузера Mozilla Firefox. Этот инструмент позволил зафиксировать поведение приложения в процессе выполнения пользовательских сценариев: от загрузки данных до построения графика и изменения параметров алгоритма. Профилирование охватывало сессию длительностью 53 секунды, в течение которой проводились переходы между этапами интерфейса, запускались вычисления через Pyodide и происходила динамическая отрисовка графика.

На временной шкале профиля (Рис. 4.4) можно наблюдать визуальные снимки интерфейса (screenshots), которые последовательно отображают ключевые состояния системы. Участки активности основного потока, отражённые в разделе Parent Process, демонстрируют кратковременные пики вычислений, особенно в области 6–7, 18–20 и 42–50 секунд, где происходили вызовы пользовательских функций и обновление визуализации. Активность по сети практически отсутствует, что подтверждает локальный характер архитектуры приложения. Нагрузка на память растёт умеренно, с постепенным накоплением и без резких скачков, что указывает на отсутствие утечек и эффективную очистку ресурсов.

Особое внимание привлекли участки в конце записи, начиная с 42 секунды, где происходило интенсивное взаимодействие с интерфейсом через ползунок параметра order. В этих точках отчётливо видны повторные вызовы отрисовки (CanvasRenderer) и аккуратное обновление DOM без лишних перерасчётов. Также не фиксировалось снижение частоты кадров или чрезмерной загрузки потока рендеринга. Общая картина профиля демонстрирует сбалансированное распределение нагрузки между рендерингом, обработкой событий и выполнением скриптов.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Временная шкала профайлера

Профилирование подтвердило, что приложение справляется с интерактивной нагрузкой, не вызывает блокировок основного потока и обеспечивает устойчивое поведение интерфейса при активных действиях пользователя. Полученные данные свидетельствуют о пригодности архитектуры для работы в реальном времени, в том числе при повторных вычислениях и визуальных обновлениях.

Оценка использования памяти в веб-приложении проводилась с целью выявления потенциальных утечек, а также анализа поведения интерфейса при повторных операциях и обновлениях графика. Исследование также проводилось с помощью встроенного профайлера Mozilla Firefox, где отслеживались изменения объёма занимаемой памяти в процессе взаимодействия с приложением.

В рамках детального анализа распределения памяти был также выполнен снимок кучи (heap snapshot), отображённый в виде карты дерева (Рис. 4.5).

Изображение выглядит как текст, снимок экрана, Прямоугольник, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Карта дерева памяти

Этот режим позволяет визуально оценить, какие типы объектов занимают основное пространство в памяти. Как видно из результатов, наибольшую долю занимают функции (Function) и DOM-узлы (DOMNode), что соответствует интерактивной природе приложения и активному использованию Canvas-графиков. Также можно выделить вклад библиотеки chart.js, загружаемой через CDN, которая отображается как отдельный скрипт с соответствующим объёмом занимаемой памяти. Отсутствие крупных неуправляемых объектов или множества повторяющихся DOM-узлов свидетельствует о корректной работе с памятью и отсутствии утечек при многократных обновлениях интерфейса.

В процессе выполнения всех действий наблюдался умеренный рост памяти в основном процессе, связанный с загрузкой данных, инициализацией Pyodide и хранением промежуточных результатов. При этом отсутствовали признаки неконтролируемого роста: объём используемой памяти стабилизировался после завершения интенсивных операций, что свидетельствует об успешном освобождении ресурсов.

Было зафиксировано, что повторные вызовы функций через Pyodide не накапливают лишние объекты в памяти. Это объясняется тем, что предыдущие результаты интерпретации перезаписываются, а связанные с ними DOM-элементы очищаются перед отрисовкой нового графика. Проверка снимков состояния памяти (heap snapshots) не выявила «висячих» DOM-узлов или неосвобождённых контекстов исполнения, что исключает типовые утечки, связанные с анонимными функциями, слушателями событий или замыканиями.

Отдельное внимание уделялось блоку визуализации: несмотря на то, что график строится повторно при изменении параметра, объём занимаемой памяти в разделе CanvasRenderer оставался стабильным, без накопления отрисованных слоёв или утечек текстур. Это подтверждает корректную работу с API отрисовки и адекватную переработку объектов визуализации при повторных вызовах.

Мониторинг использования памяти подтвердил, что реализация интерфейса и логики обработки данных не приводит к накоплению «мусорных» объектов, а используемые библиотеки (Chart.js и Pyodide) работают предсказуемо и без создания скрытых ссылок. Это позволяет считать систему устойчивой к длительному взаимодействию с пользователем и масштабируемой при увеличении объёма данных или количества вычислений.

Для оценки корректности работы событийной модели и своевременного освобождения ресурсов была проведена проверка активности слушателей событий и реактивного обновления компонентов интерфейса. Основное внимание уделялось динамическим участкам приложения – в частности, обработке кликов, переключению шагов и взаимодействию с ползунком параметра order.

Проверка выполнялась с использованием вкладки Inspector в Firefox Developer Tools. Перейдя к интересующему DOM-элементу (например, кнопке или слайдеру), можно открыть панель Event Listeners, в которой отображаются все активные обработчики, привязанные к данному элементу. При переключении между шагами проверялось, что слушатели не дублируются и не остаются «подвешенными» на скрытых элементах, что потенциально могло бы привести к утечкам памяти или некорректной логике исполнения.

Результаты анализа показали, что приложение использует прямую привязку событий к элементам с помощью addEventListener, а также очищает и повторно инициализирует интерфейс корректно при возврате к предыдущим шагам. Не было выявлено дублирующихся обработчиков или следов множественных подписок на одни и те же действия. Также отсутствуют признаки утечки контекстов замыканий, что могло бы указывать на неосвобождаемые ссылки в памяти. Пример EventListener на кнопке приведён на Рис. 4.6.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. . – Пример обработчика события click в Inspector

Отдельно оценивалась реактивность интерфейса (способность мгновенно откликаться на действия пользователя). Наиболее чувствительным элементом является ползунок для изменения параметра order, при перемещении которого вызывается функция на Python и перерисовывается график. Несмотря на потенциальную нагрузку, отклик интерфейса остаётся стабильным, а время реакции минимальным. Это свидетельствует о том, что обработка изменений реализована эффективно: используется событие input или change, обновления не накапливаются в очереди, а вызываются по мере необходимости.

Проверка слушателей событий и реактивности подтверждает, что интерфейс приложения построен с учётом требований к масштабируемости и устойчивости. Реализация не допускает накопления избыточных подписок и обеспечивает мгновенный отклик на пользовательские действия без задержек или конфликтов в логике обработки событий.

Таким образом, в результате проведённого динамического анализа были подтверждены ключевые характеристики стабильной и эффективной работы клиентского интерфейса. Приложение демонстрирует устойчивое поведение при выполнении последовательных действий пользователя: загрузке файлов, переключении между шагами, запуске пользовательских алгоритмов и динамической отрисовке графиков. Профилирование производительности показало, что основная нагрузка распределяется равномерно, время отклика интерфейса остаётся минимальным даже при повторных вычислениях, а процессы отрисовки выполняются без блокировки главного потока исполнения.

## Выводы по результатам тестирования и анализа

В рамках четвёртой главы были проведены комплексные мероприятия по оценке надёжности, устойчивости и корректности работы разработанного клиентского веб-приложения. Исследование включало как формализованные подходы, такие как статический и модульный анализ кода, так и наблюдение за поведением системы в условиях реального выполнения.

На этапе статического анализа средствами ESLint были выявлены и устранены базовые синтаксические конфликты, связанные с использованием модульной структуры кода. Дополнительно была обеспечена единообразная структура скриптов, что повысило читаемость и упростило поддержку проекта.

В процессе модульного тестирования с использованием среды Jest и библиотеки jsdom были реализованы тесты, охватывающие ключевые участки функциональности: загрузку файлов, переключение между шагами, отрисовку графика и обработку пользовательских параметров. Все тесты прошли успешно, что подтвердило устойчивость интерфейсной логики и корректность DOM-взаимодействий.

Далее был проведён динамический анализ поведения приложения, охватывающий профилирование производительности, контроль за использованием памяти, мониторинг активности слушателей событий и реактивность интерфейса. В результате профилирования было установлено, что приложение демонстрирует стабильную работу как при единичных действиях, так и в условиях интенсивного пользовательского взаимодействия. Используемые архитектурные решения, включая повторное использование DOM-элементов, ручное управление событиями и локальное выполнение кода на стороне клиента через Pyodide, не приводят к деградации производительности, утечкам памяти или сбоям интерфейса.

Таким образом, проведённое тестирование и анализ подтверждают, что разработанное приложение соответствует требованиям к стабильной, масштабируемой и интерактивной системе. Выбранные технологии и методы реализации обеспечивают достаточный уровень надёжности и готовы к дальнейшему расширению функциональности.

# ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы была решена задача разработки системы ассистированного преобразования сигналов, предназначенной для прикладных задач, в которых невозможно определить формализованный критерий качества результата. Такой подход особенно актуален при решении некорректных задач, требующих экспертной оценки и визуального анализа.

Был проведён детальный обзор предметной области, в том числе анализ классов некорректных задач и существующих решений (MATLAB, OriginLab, Chromeleon). Обоснована необходимость разработки специализированного инструмента, предоставляющего интерактивные средства управления и визуальной настройки параметров обработки данных.

На основе анализа был предложен технологический стек, включающий HTML, JavaScript, Chart.js и Pyodide. Проведён выбор архитектуры одностраничного приложения с полной реализацией вычислений и визуализации на стороне клиента без обращения к серверной части.

Разработана программная система, включающая:

* загрузку пользовательских CSV-файлов и Python-скриптов;
* автоматическое извлечение функций и параметров;
* реализацию механизма визуального управления параметрами;
* немедленную отрисовку результатов с помощью Chart.js;
* безопасное исполнение пользовательского кода в среде Pyodide.

В ходе тестирования проведён статический анализ кода с использованием ESLint, модульное тестирование при помощи Jest и jsdom, а также динамическое профилирование с помощью Firefox Developer Tools. Анализ показал устойчивую работу интерфейса, корректное освобождение памяти и отсутствие утечек. Тесты подтвердили стабильность ключевых сценариев.

Проведённый динамический анализ выявил отсутствие существенных узких мест по производительности и подтвердил реактивность интерфейса при повторных вычислениях и визуальных обновлениях. Слушатели событий обрабатываются корректно, не дублируются и не вызывают утечек.

Полученные результаты демонстрируют практическую применимость разработанной системы для широкого спектра пользовательских задач, где экспертное восприятие играет определяющую роль. Универсальность архитектуры позволяет расширять функциональность без переработки базовых модулей.

Перспективы дальнейшего развития системы включают переход к клиент-серверной архитектуре с выносом ресурсоёмких вычислений на серверную сторону, что обеспечит обработку больших объёмов данных и расширит функциональные возможности. Также планируется внедрение интеллектуального помощника, использующего накопленную историю взаимодействий пользователя для автоматического подбора параметров и адаптации интерфейса под индивидуальные сценарии работы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы: учебник – М.: Лаборатория знаний, 2020. – 636 с.
2. Грешилов А.А. Некорректные задачи цифровой обработки информации и сигналов: учебное пособие – М.: Логос, 2009. – 360 с.
3. Документация Chart.js [Электронный ресурс]. – URL: https://www.chartjs.org/docs/latest/ (дата обращения: 12.05.2025).
4. Документация CSS [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Web/CSS (дата обращения: 10.05.2025).
5. Документация D3.js [Электронный ресурс]. – URL: https://d3js.org/ (дата обращения: 11.05.2025).
6. Документация ESLint [Электронный ресурс]. – URL: https://eslint.org/docs/latest/ (дата обращения: 21.05.2025).
7. Документация Firefox DevTools [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Tools (дата обращения: 21.05.2025).
8. Документация HTML [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Web/HTML (дата обращения: 10.05.2025).
9. Документация JavaScript [Электронный ресурс]. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата обращения: 14.05.2025).
10. Документация Jest [Электронный ресурс]. – URL: https://jestjs.io/docs/getting-started (дата обращения: 21.05.2025).
11. Документация Matplotlib [Электронный ресурс]. – URL: https://matplotlib.org/stable/index.html (дата обращения: 12.05.2025).
12. Документация MDN Web API [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Web/API (дата обращения: 12.05.2025).
13. Документация NumPy [Электронный ресурс]. – URL: https://numpy.org/doc/stable/ (дата обращения: 12.05.2025).
14. Документация OriginLab [Электронный ресурс]. – URL: https://www.originlab.com/index.aspx?go=support/DocumentationAndHelpCenter (дата обращения: 11.05.2025).
15. Документация Pandas [Электронный ресурс]. – URL: https://pandas.pydata.org/docs/ (дата обращения: 12.05.2025).
16. Документация Plotly.js [Электронный ресурс]. – URL: https://plotly.com/javascript/ (дата обращения: 11.05.2025).
17. Документация Pyodide [Электронный ресурс]. – URL: https://pyodide.org/en/stable/ (дата обращения: 12.05.2025).
18. Документация React [Электронный ресурс]. – URL: https://react.dev/reference/react (дата обращения: 10.05.2025).
19. Документация REST API [Электронный ресурс]. – URL: https://docs.github.com/ru/rest?apiVersion=2022-11-28 (дата обращения: 12.05.2025).
20. Документация Sass [Электронный ресурс]. – URL: https://sass-lang.com/documentation/ (дата обращения: 14.05.2025).
21. Документация scikit-learn [Электронный ресурс]. – URL: https://scikit-learn.org/stable/ (дата обращения: 12.05.2025).
22. Документация SciPy [Электронный ресурс]. – URL: https://docs.scipy.org/doc/scipy/ (дата обращения: 12.05.2025).
23. Документация MATLAB Signal Processing Toolbox [Электронный ресурс]. – URL: https://www.mathworks.com/help/signal/index.html (дата обращения: 11.05.2025).
24. Документация WebAssembly [Электронный ресурс]. – URL: https://webassembly.org/docs/faq/ (дата обращения: 12.05.2025).
25. Кройц М.А., Морозов О.А. Методы решения обратных задач в цифровой обработке сигналов: практикум – Нижний Новгород: Нижегородский госуниверситет, 2020 – 26 с.
26. Официальный сайт браузера Google Chrome [Электронный ресурс]. – URL: https://www.google.com/intl/ru/chrome/ (дата обращения: 16.05.2025).
27. Официальный сайт браузера Mozilla Firefox [Электронный ресурс]. – URL: https://www.mozilla.org/ru/firefox/new/ (дата обращения: 16.05.2025).
28. Официальный сайт браузера Yandex Browser [Электронный ресурс]. – URL: https://browser.yandex.ru/ (дата обращения: 16.05.2025).
29. Официальный сайт ПО «Chromeleon» [Электронный ресурс]. – URL: https://www.thermofisher.com/order/catalog/product/CHROMELEON7 (дата обращения: 11.05.2025).
30. Севастьянов А.А., Харинцев С.С., Салахов М.Х. Нейросетевая регуляризация решения обратных некорректных задач прикладной спектроскопии // Исследовано в России. – 2003. – Т.8. – С. 2254-2266.
31. Тихонов А.Н., Арсенин В.Я. Методы решения некорректных задач: учебник – М.: Наука, 1979. – 288 с.
32. Цымблер М.Л., Юртин А.А. Восстановление пропущенных значений временного ряда на основе совместного применения аналитических алгоритмов и нейронных сетей // Вычислительные методы и программирование. – 2023. – Т.24, В.3 – С. 243-259.
33. Aydin M. Torkabadi, Esam M.A. Hussein. Assessing the credibility of the solutions of incomplete-data inverse problems: Physics Open, Volume 7, 2021.
34. Emery Bosten, Kai Chen, Mario Hellings, Deirdre Cabooter. Artificial intelligence for method development in liquid chromatography: TrAC Trends in Analytical Chemistry, Volume 192, 2025.
35. M. Bertero, P.Boccacci, Christine De Mol. Introduction to Inverse Problems in Imaging: 2nd Edition, Boca Raton, CRC Press, 2021. – 358 p.
36. Richard C. Aster, Brian Borchers, Clifford H. Thurber. Parameter Estimation and Inverse Problems: 3rd Edition, Elsevier, 2018. – 404 p.

# Приложение 1

Полный код программы

index.html

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="UTF-8" />

        <meta

            name="viewport"

            content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0"

        />

        <meta http-equiv="X-UA-Compatible" content="ie=edge" />

        <link rel="stylesheet" href="./styles/styles.css" />

        <title>Diploma</title>

    </head>

    <body>

        <header class="header">

            <div class="header\_\_inner container">

                <div class="header\_\_top">

                    <a class="header\_\_logo logo" href="/">

                        <img

                            class="logo\_\_image"

                            src="./images/logo.svg"

                            alt="Logo"

                            width="75"

                            height="75"

                            loading="lazy"

                        />

                    </a>

                    <h1 class="header\_\_logo-text">

                        Система ассистированного преобразования сигналов

                    </h1>

                </div>

                <nav class="header\_\_menu">

                    <ul class="header\_\_menu-list">

                        <li class="header\_\_menu-item">

                            <a class="header\_\_menu-link" href="/">Инструкция</a>

                        </li>

                        <li class="header\_\_menu-item">

                            <a class="header\_\_menu-link" href="/">Демо-режим</a>

                        </li>

                        <li class="header\_\_menu-item">

                            <a class="header\_\_menu-link" href="/">Авторы</a>

                        </li>

                    </ul>

                </nav>

            </div>

        </header>

        <main class="content">

            <section class="section step step--active container" id="step1">

                <div class="section\_\_body">

                    <div class="data">

                        <h2 class="data\_\_title">Загрузка данных</h2>

                        <p class="data\_\_description">

                            Загрузите данные (поддерживаемые форматы:

                            <strong>.csv, .jpg, .png</strong>)

                        </p>

                        <label class="data\_\_button button" for="fileInput">

                            Выбрать файл

                        </label>

                        <input

                            type="file"

                            id="fileInput"

                            accept=".csv, .jpg, .png"

                            style="display: none"

                        />

                        <p class="data\_\_button-description">

                            или перетащите файл сюда

                        </p>

                    </div>

                    <div id="chartContainer" class="data\_\_chart hidden"></div>

                </div>

            </section>

            <!-- Шаг 2: Загрузка скрипта обработки -->

            <section id="step2" class="section step container">

                <div class="section\_\_body step\_\_body">

                    <div class="data">

                        <h2 class="data\_\_title">

                            Загрузка скрипта обработки данных

                        </h2>

                        <p class="data\_\_description">

                            Загрузите программу (поддерживаемый формат:

                            <strong>.py</strong>)

                        </p>

                        <label class="data\_\_button button" for="scriptInput">

                            Выбрать файл

                        </label>

                        <input

                            type="file"

                            id="scriptInput"

                            accept=".py"

                            style="display: none"

                        />

                        <p class="data\_\_button-description">

                            или перетащите файл сюда

                        </p>

                    </div>

                    <!-- Кнопка "Предыдущий шаг" -->

                    <div class="step\_\_navigation">

                        <button class="button" id="backToStep1">

                            Предыдущий шаг

                        </button>

                    </div>

                </div>

            </section>

            <!-- Шаг 3: Выбор функции и параметров -->

            <section id="step3" class="section step container">

                <div class="section\_\_body step\_\_body">

                    <div class="data">

                        <h2 class="data\_\_title">Выбор функции и параметров</h2>

                        <p class="data\_\_description">

                            Выберите функцию, а также задайте начальные и

                            конечные значения входным параметрам

                        </p>

                        <div class="function-container">

                            <form id="functionForm" class="data\_\_form">

                                <!-- Функции появятся здесь -->

                            </form>

                            <div class="step\_\_navigation">

                                <button class="button" id="backToStep2">

                                    Предыдущий шаг

                                </button>

                                <button

                                    class="button"

                                    id="toStep4"

                                    style="margin-left: auto"

                                >

                                    Следующий шаг

                                </button>

                            </div>

                        </div>

                    </div>

                </div>

            </section>

            <section id="step4" class="section step container">

                <div class="section\_\_body step\_\_body">

                    <div class="data">

                        <h2 class="data\_\_title">Результат</h2>

                        <div class="data\_\_description">

                            <div

                                id="orderControl"

                                style="margin-bottom: 1rem"

                            ></div>

                            <div id="chartArea" class="data\_\_chart"></div>

                        </div>

                    </div>

                    <div class="step\_\_navigation">

                        <button class="button" id="backToStep3">

                            Предыдущий шаг

                        </button>

                    </div>

                </div>

            </section>

            <aside class="sidebar" id="sidebar">

                <h3 class="sidebar\_\_title">Файл загружен!</h3>

                <p class="sidebar\_\_filename" id="fileNameDisplay"></p>

                <div class="sidebar\_\_actions">

                    <button

                        class="sidebar\_\_actions--plot button"

                        id="plotButton"

                    >

                        Показать график

                    </button>

                    <button class="sidebar\_\_actions--next button" type="button">

                        Следующий шаг

                    </button>

                </div>

            </aside>

        </main>

        <footer class="footer">

            <div class="footer\_\_inner container">© 2025</div>

        </footer>

        <!-- Скрипты -->

        <script type="module" src="./scripts/sidebar.js"></script>

        <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

        <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-zoom@2.0.1/dist/chartjs-plugin-zoom.min.js"></script>

        <script src="https://cdn.jsdelivr.net/pyodide/v0.23.4/full/pyodide.js"></script>

    </body>

</html>

styles.css

@charset "UTF-8";

/\*\*

  Нормализация блочной модели

 \*/

\*,

::before,

::after {

  box-sizing: border-box;

}

/\*\*

  Убираем внутренние отступы слева тегам списков,

  у которых есть атрибут class

 \*/

:where(ul, ol):where([class]) {

  padding-left: 0;

}

/\*\*

  Убираем внешние отступы body и двум другим тегам,

  у которых есть атрибут class

 \*/

body,

:where(blockquote, figure):where([class]) {

  margin: 0;

}

/\*\*

  Убираем внешние отступы вертикали нужным тегам,

  у которых есть атрибут class

 \*/

:where(h1,

h2,

h3,

h4,

h5,

h6,

p,

ul,

ol,

dl):where([class]) {

  margin-block: 0;

}

:where(dd[class]) {

  margin-left: 0;

}

:where(fieldset[class]) {

  margin-left: 0;

  padding: 0;

  border: none;

}

/\*\*

  Убираем стандартный маркер маркированному списку,

  у которого есть атрибут class

 \*/

:where(ul[class]) {

  list-style: none;

}

:where(address[class]) {

  font-style: normal;

}

/\*\*

  Обнуляем вертикальные внешние отступы параграфа,

  объявляем локальную переменную для внешнего отступа вниз,

  чтобы избежать взаимодействие с более сложным селектором

 \*/

p {

  --paragraphMarginBottom: 24px;

  margin-block: 0;

}

/\*\*

  Внешний отступ вниз для параграфа без атрибута class,

  который расположен не последним среди своих соседних элементов

 \*/

p:where(:not([class]):not(:last-child)) {

  margin-bottom: var(--paragraphMarginBottom);

}

/\*\*

  Упрощаем работу с изображениями и видео

 \*/

img,

video {

  display: block;

  max-width: 100%;

  height: auto;

}

/\*\*

  Наследуем свойства шрифт для полей ввода

 \*/

input,

textarea,

select,

button {

  font: inherit;

}

html {

  /\*\*

    Пригодится в большинстве ситуаций

    (когда, например, нужно будет "прижать" футер к низу сайта)

   \*/

  height: 100%;

  /\*\*

    Убираем скачок интерфейса по горизонтали

    при появлении / исчезновении скроллбара

   \*/

  scrollbar-gutter: stable;

}

/\*\*

  Плавный скролл

 \*/

html,

:has(:target) {

  scroll-behavior: smooth;

}

body {

  /\*\*

    Пригодится в большинстве ситуаций

    (когда, например, нужно будет "прижать" футер к низу сайта)

   \*/

  min-height: 100%;

  /\*\*

    Унифицированный интерлиньяж

   \*/

  line-height: 1.5;

}

/\*\*

  Нормализация высоты элемента ссылки при его инспектировании в DevTools

 \*/

a:where([class]) {

  display: inline-flex;

}

/\*\*

  Курсор-рука при наведении на элемент

 \*/

button,

label {

  cursor: pointer;

}

/\*\*

  Приводим к единому цвету svg-элементы

  (за исключением тех, у которых уже указан

  атрибут fill со значением 'none' или начинается с 'url')

 \*/

:where([fill]:not([fill=none],

[fill^=url])) {

  fill: currentColor;

}

/\*\*

  Приводим к единому цвету svg-элементы

  (за исключением тех, у которых уже указан

  атрибут stroke со значением 'none')

 \*/

:where([stroke]:not([stroke=none],

[stroke^=url])) {

  stroke: currentColor;

}

/\*\*

  Чиним баг задержки смены цвета при взаимодействии с svg-элементами

 \*/

svg \* {

  transition-property: fill, stroke;

}

/\*\*

  Приведение рамок таблиц в классический 'collapse' вид

 \*/

:where(table) {

  border-collapse: collapse;

  border-color: currentColor;

}

/\*\*

  Удаляем все анимации и переходы для людей,

  которые предпочитают их не использовать

 \*/

@media (prefers-reduced-motion: reduce) {

  \*,

  ::before,

  ::after {

    animation-duration: 0.01ms !important;

    animation-iteration-count: 1 !important;

    transition-duration: 0.01ms !important;

    scroll-behavior: auto !important;

  }

}

@font-face {

  font-family: "Ubuntu";

  src: url("../fonts/Ubuntu-Light.woff2") format("woff2");

  font-weight: 300;

  font-style: normal;

  font-display: swap;

}

@font-face {

  font-family: "Ubuntu";

  src: url("../fonts/Ubuntu-Regular.woff2") format("woff2");

  font-weight: 400;

  font-style: normal;

  font-display: swap;

}

@font-face {

  font-family: "Ubuntu";

  src: url("../fonts/Ubuntu-Medium.woff2") format("woff2");

  font-weight: 500;

  font-style: normal;

  font-display: swap;

}

@font-face {

  font-family: "Ubuntu";

  src: url("../fonts/Ubuntu-Bold.woff2") format("woff2");

  font-weight: 700;

  font-style: normal;

  font-display: swap;

}

/\* Миксины \*/

/\*tablet-above\*/

/\*tablet\*/

/\*mobile-above\*/

/\*mobile\*/

:root {

  --color-dark: #000000;

  --color-light: #FFFFFF;

  --color-accent: #373062;

  --color-alternate: #724796;

  --border: 1px solid var(--color-dark);

  --border-radius: 14px;

  /\*shadow\*/

  --font-family-base: "Ubuntu", sans-serif;

  --container-width: 1240px;

  --container-padding-x: 20px;

  --transition-duration: 0.2s;

}

/\* Утилитарные функции (контейнер, visually-hidden и т.д.) \*/

.container {

  max-width: calc(var(--container-width) + var(--container-padding-x) \* 2);

  margin-inline: auto;

  padding-inline: var(--container-padding-x);

}

.visually-hidden {

  position: absolute !important;

  width: 1px !important;

  height: 1px !important;

  margin: -1px !important;

  border: 0 !important;

  padding: 0 !important;

  white-space: nowrap !important;

  -webkit-clip-path: inset(100%) !important;

          clip-path: inset(100%) !important;

  clip: rect(0 0 0 0) !important;

  overflow: hidden !important;

}

.hidden {

  display: none !important;

}

/\*hidden-mobile\*/

/\*visible-mobile\*/

/\* Глобальные стили для элементов html, body, ссылок, кнопок, полей ввода и т.д. \*/

body {

  height: 100%;

  display: flex;

  flex-direction: column;

  font-family: var(--font-family-base);

  font-weight: 400;

  font-size: 20px;

  line-height: 1.28;

  color: var(--color-dark);

  background-color: var(--color-light);

}

a,

button,

input,

svg \* {

  transition-duration: var(--transition-duration);

}

a {

  text-decoration: none;

  color: inherit;

  font-family: inherit;

}

h1, h2, h3, h4 {

  font-weight: 500;

}

.button {

  display: inline-flex;

  justify-content: center;

  align-items: center;

  height: 75px;

  padding-inline: 50px;

  font-size: 24px;

  color: var(--color-light);

  background-color: var(--color-accent);

  border: var(--border);

  border-radius: 14px;

}

@media (hover: hover) {

  .button:hover {

    background-color: var(--color-dark);

  }

}

@media (hover: none) {

  .button:active {

    background-color: var(--color-dark);

  }

}

.header {

  padding-top: 40px;

  padding-bottom: 20px;

  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);

  z-index: 100;

  position: relative;

}

.header\_\_inner {

  display: flex;

  flex-direction: column;

  row-gap: 20px;

}

.header\_\_top {

  display: flex;

  justify-content: center;

  align-items: center;

  -moz-column-gap: 40px;

       column-gap: 40px;

}

.header\_\_menu-list {

  display: flex;

  justify-content: center;

  align-items: center;

  -moz-column-gap: 40px;

       column-gap: 40px;

}

.header\_\_menu-link {

  position: relative;

  display: inline-flex;

  align-items: center;

  height: 48px;

}

@media (hover: hover) {

  .header\_\_menu-link:hover::after {

    width: 100%;

  }

}

@media (hover: none) {

  .header\_\_menu-link:active::after {

    width: 100%;

  }

}

.header\_\_menu-link::after {

  content: "";

  position: absolute;

  top: 100%;

  left: 50%;

  translate: -50%;

  width: 0;

  height: 2px;

  background-color: currentColor;

  transition-duration: inherit;

}

main {

  flex: 1 0 auto;

}

.section.container {

  flex: 1;

  transition: all 0.3s ease;

}

.content {

  display: flex;

  flex-direction: row;

  align-items: stretch;

  position: relative;

}

.section,

.step {

  flex: 1;

  transition: all 0.3s ease;

  min-width: 0;

}

.step {

  display: none;

  width: 100%;

}

.step--active {

  display: block;

}

.step\_\_body {

  display: flex;

  flex-direction: column;

  justify-content: center;

  align-items: center;

  height: 100%;

}

.step\_\_content {

  display: flex;

  flex-direction: column;

  flex-grow: 1;

}

.step\_\_navigation {

  margin-top: auto;

  display: flex;

  justify-content: center;

  padding: 2rem 0;

}

#backToStep1 {

  font-size: 20px;

  height: 60px;

}

#backToStep2 {

  font-size: 20px;

  height: 60px;

}

#backToStep3 {

  font-size: 20px;

  height: 60px;

}

#toStep4 {

  font-size: 20px;

  height: 60px;

}

.section {

  padding-top: 40px;

}

#chartArea {

  width: 100%;

  height: 600px;

  display: flex;

  justify-content: center;

}

#chartArea canvas {

  width: 100% !important;

  height: 100% !important;

}

.data {

  display: flex;

  flex-direction: column;

  align-items: center;

  flex-grow: 1;

}

.data\_\_title {

  font-size: 32px;

  margin-bottom: 10px;

}

.data\_\_description {

  margin-bottom: 30px;

  font-weight: 300;

}

.data\_\_button {

  margin-bottom: 5px;

}

.data\_\_button-description {

  justify-content: center;

  align-items: center;

  font-size: 18px;

  font-weight: 300;

}

.data\_\_chart {

  margin-top: 20px;

}

.sidebar {

  width: 0;

  overflow: hidden;

  transition: width 0.3s ease, padding 0.3s ease;

  padding: 0;

  background-color: #fff;

  border-left: 1px solid #ddd;

  box-shadow: -4px 0 10px rgba(0, 0, 0, 0.1);

  white-space: nowrap;

  z-index: 50;

  display: flex;

  flex-direction: column;

  justify-content: space-between;

}

.sidebar--visible {

  width: 350px;

  padding: 20px;

}

.sidebar\_\_title {

  text-align: center;

  margin-bottom: 20px;

}

.sidebar\_\_filename {

  word-break: break-word;

  white-space: normal;

}

.sidebar\_\_actions {

  display: flex;

  flex-direction: column;

  gap: 12px;

  margin-top: auto;

}

.sidebar\_\_actions--plot {

  font-size: 20px;

  height: 60px;

  background-color: var(--color-light);

  color: var(--color-dark);

}

.sidebar\_\_actions--plot:hover {

  background-color: var(--color-accent);

  color: var(--color-light);

}

.sidebar\_\_actions--next {

  font-size: 20px;

  height: 60px;

}

.sidebar #plotButton {

  display: none;

}

.footer {

  padding-block: 20px;

  color: var(--color-light);

  background-color: var(--color-accent);

  box-shadow: 0 -4px 10px rgba(0, 0, 0, 0.1);

}/\*# sourceMappingURL=styles.css.map \*/

chart.js

// chart.js

function parseCSV(text) {

    return text

        .trim()

        .split("\n")

        .map((line) => line.split(",").map((cell) => cell.trim()));

}

export async function renderChart(container, file) {

    container.innerHTML = "";

    const canvas = document.createElement("canvas");

    container.appendChild(canvas);

    const text = await file.text();

    const rows = parseCSV(text);

    if (rows.length < 2 || rows[0].length < 2) {

        container.innerHTML =

            "<p>Недостаточно данных для построения графика.</p>";

        return;

    }

    const headers = rows[0];

    const labels = rows.slice(1).map((row) => row[0]);

    const datasets = headers.slice(1).map((colName, colIndex) => {

        const data = rows.slice(1).map((row) => parseFloat(row[colIndex + 1]));

        return {

            label: colName,

            data,

            fill: false,

            borderColor: "#373062",

            tension: 0.1,

        };

    });

    const ctx = canvas.getContext("2d");

    new Chart(ctx, {

        type: "line",

        data: {

            labels,

            datasets,

        },

        options: {

            responsive: true,

            maintainAspectRatio: true,

            plugins: {

                zoom: {

                    zoom: {

                        wheel: {

                            enabled: true,

                        },

                        pinch: {

                            enabled: true,

                        },

                        mode: "x",

                    },

                    pan: {

                        enabled: false,

                    },

                },

            },

        },

    });

    container.style.height = "600px";

}

sidebar.js

import { renderChart } from "./chart.js";

const fileInput = document.getElementById("fileInput");

const scriptInput = document.getElementById("scriptInput");

const sidebar = document.getElementById("sidebar");

const fileNameDisplay = document.getElementById("fileNameDisplay");

const plotButton = document.getElementById("plotButton");

const chooseFileButton = document.querySelector(".data\_\_button");

const dataBlock = document.querySelector("#step1 .data");

const chartContainer = document.getElementById("chartContainer");

const nextStepButton = document.querySelector(".sidebar\_\_actions--next");

const backToStep1Button = document.getElementById("backToStep1");

// === Шаг 3 ===

const step3 = document.getElementById("step3");

const functionForm = document.getElementById("functionForm");

const backToStep2 = document.getElementById("backToStep2");

let chartVisible = false;

let uploadedImage = null;

let uploadedFile = null;

let pyContent = "";

// === ЗАГРУЗКА CSV/JPG/PNG ===

chooseFileButton.addEventListener("click", () => {

    fileInput.click();

});

fileInput.addEventListener("change", (event) => {

    const file = event.target.files[0];

    if (!file) return;

    const validExtensions = ["csv", "jpg", "png"];

    const fileExt = file.name.split(".").pop().toLowerCase();

    if (!validExtensions.includes(fileExt)) {

        alert("Неподдерживаемый формат файла");

        return;

    }

    window.uploadedFile = file;

    if (fileExt === "jpg" || fileExt === "png") {

        const url = URL.createObjectURL(file);

        uploadedImage = {

            name: file.name,

            url: url,

            file: file,

        };

    } else {

        uploadedImage = null;

    }

    fileNameDisplay.textContent = file.name;

    sidebar.classList.add("sidebar--visible");

    if (fileExt === "csv") {

        plotButton.style.display = "block";

    } else {

        plotButton.style.display = "none";

    }

    dataBlock.classList.remove("hidden");

    chartContainer.classList.add("hidden");

    chartContainer.innerHTML = "";

    plotButton.textContent = "Показать график";

    chartVisible = false;

    // ДОБАВЛЯЕМ ЭТО

    if (fileExt === "jpg" || fileExt === "png") {

        const url = URL.createObjectURL(file);

        uploadedImage = {

            name: file.name,

            url: url,

            file: file,

        };

    } else {

        uploadedImage = null;

    }

});

// === ГРАФИК CSV ===

plotButton.addEventListener("click", async () => {

    chartVisible = !chartVisible;

    if (chartVisible) {

        dataBlock.classList.add("hidden");

        chartContainer.classList.remove("hidden");

        await renderChart(chartContainer, window.uploadedFile);

        plotButton.textContent = "Скрыть график";

    } else {

        dataBlock.classList.remove("hidden");

        chartContainer.classList.add("hidden");

        chartContainer.innerHTML = "";

        plotButton.textContent = "Показать график";

    }

});

// === ПЕРЕХОД НА ВТОРОЙ ШАГ ===

nextStepButton.addEventListener("click", () => {

    const step2 = document.getElementById("step2");

    // Если мы ещё на шаге 1 — просто переходим к шагу 2

    if (!step2.classList.contains("step--active")) {

        document.getElementById("step1").classList.remove("step--active");

        step2.classList.add("step--active");

        sidebar.classList.remove("sidebar--visible");

        dataBlock.classList.remove("hidden");

        chartContainer.classList.add("hidden");

        chartContainer.innerHTML = "";

        plotButton.textContent = "Показать график";

        chartVisible = false;

        return;

    }

    // === Если мы уже на шаге 2 — переходим на шаг 3 ===

    step2.classList.remove("step--active");

    step3.classList.add("step--active");

    sidebar.classList.remove("sidebar--visible");

    functionForm.innerHTML = "";

    const functions = parsePythonFunctions(window.pyContent);

    if (functions.length === 0) {

        functionForm.innerHTML = "<p>Функции не найдены.</p>";

        return;

    }

    functions.forEach((fn, index) => {

        const fnBlock = document.createElement("div");

        fnBlock.className = "function-block";

        fnBlock.style.marginBottom = "1rem";

        const radio = document.createElement("input");

        radio.type = "radio";

        radio.name = "selectedFunction";

        radio.value = fn.name;

        radio.id = `fn-${index}`;

        if (index === 0) radio.checked = true;

        const label = document.createElement("label");

        label.htmlFor = `fn-${index}`;

        label.textContent = fn.name;

        label.style.fontWeight = "bold";

        label.style.marginLeft = "0.5rem";

        fnBlock.appendChild(radio);

        fnBlock.appendChild(label);

        fn.args.forEach((arg, idx) => {

            const argGroup = document.createElement("div");

            argGroup.style.marginTop = "0.5rem";

            argGroup.style.display = "flex";

            argGroup.style.alignItems = "center";

            argGroup.style.gap = "0.5rem";

            if (idx === 0) {

                const label = document.createElement("label");

                label.textContent = `Параметр "${arg}":`;

                const select = document.createElement("select");

                select.name = `${fn.name}\_${arg}`;

                select.id = `${fn.name}\_${arg}`;

                if (

                    window.uploadedFile &&

                    window.uploadedFile.name.endsWith(".csv")

                ) {

                    const option = document.createElement("option");

                    option.value = window.uploadedFile.name;

                    option.textContent = window.uploadedFile.name;

                    select.appendChild(option);

                } else {

                    const option = document.createElement("option");

                    option.value = "";

                    option.textContent = "Нет загруженного CSV-файла";

                    select.appendChild(option);

                    select.disabled = true;

                }

                argGroup.appendChild(label);

                argGroup.appendChild(select);

            } else {

                // Остальные параметры — чекбокс + min/max

                const checkbox = document.createElement("input");

                checkbox.type = "checkbox";

                checkbox.name = `${fn.name}\_${arg}\_active`;

                checkbox.id = `${fn.name}\_${arg}\_active`;

                checkbox.checked = true;

                const checkboxLabel = document.createElement("label");

                checkboxLabel.htmlFor = checkbox.id;

                checkboxLabel.textContent = `Параметр "${arg}":`;

                const minInput = document.createElement("input");

                minInput.id = `${fn.name}\_${arg}\_min`;

                minInput.type = "number";

                minInput.name = `${fn.name}\_${arg}\_min`;

                minInput.placeholder = "Мин";

                minInput.style.width = "70px";

                const maxInput = document.createElement("input");

                maxInput.id = `${fn.name}\_${arg}\_max`;

                maxInput.type = "number";

                maxInput.name = `${fn.name}\_${arg}\_max`;

                maxInput.placeholder = "Макс";

                maxInput.style.width = "70px";

                argGroup.appendChild(checkbox);

                argGroup.appendChild(checkboxLabel);

                argGroup.appendChild(minInput);

                argGroup.appendChild(maxInput);

            }

            fnBlock.appendChild(argGroup);

        });

        functionForm.appendChild(fnBlock);

    });

});

// === НАЗАД НА ШАГ 1 ===

backToStep1Button.addEventListener("click", () => {

    document.getElementById("step2").classList.remove("step--active");

    document.getElementById("step1").classList.add("step--active");

    sidebar.classList.add("sidebar--visible");

    dataBlock.classList.remove("hidden");

    chartContainer.classList.add("hidden");

    chartContainer.innerHTML = "";

    plotButton.textContent = "Показать график";

    chartVisible = false;

});

// === НАЗАД НА ШАГ 2 ===

backToStep2.addEventListener("click", () => {

    step3.classList.remove("step--active");

    document.getElementById("step2").classList.add("step--active");

});

// === ЗАГРУЗКА .py ===

scriptInput.addEventListener("change", (event) => {

    const file = event.target.files[0];

    if (!file) return;

    const fileExt = file.name.split(".").pop().toLowerCase();

    if (fileExt !== "py") {

        alert("Загрузите файл в формате .py");

        return;

    }

    fileNameDisplay.textContent = file.name;

    sidebar.classList.add("sidebar--visible");

    const reader = new FileReader();

    reader.onload = function (e) {

        window.pyContent = e.target.result;

    };

    reader.readAsText(file);

});

// === Парсинг функций ===

function parsePythonFunctions(code) {

    const lines = code.split("\n");

    const functionRegex = /^\s\*def\s+([a-zA-Z\_][a-zA-Z0-9\_]\*)\((.\*?)\):/;

    const functions = [];

    lines.forEach((line) => {

        const match = line.match(functionRegex);

        if (match) {

            const name = match[1];

            const argsRaw = match[2].split(",").map((a) => a.trim());

            // Убираем self и параметры со значением по умолчанию

            const args = argsRaw.filter(

                (a) => a && !a.includes("=") && a !== "self"

            );

            // Добавляем только если есть хотя бы один аргумент

            if (args.length > 0) {

                functions.push({ name, args });

            }

        }

    });

    return functions;

}

// Назад со 4 на 3

document.getElementById("backToStep3")?.addEventListener("click", () => {

    document.getElementById("step4")?.classList.remove("step--active");

    document.getElementById("step3")?.classList.add("step--active");

});

let pyodide = null;

let chartInstance = null;

async function initPyodide() {

    if (!pyodide) {

        pyodide = await loadPyodide({

            indexURL: "https://cdn.jsdelivr.net/pyodide/v0.23.4/full/",

        });

    }

    return pyodide;

}

document.getElementById("toStep4").addEventListener("click", () => {

    const selectedFn = document.querySelector(

        "input[name='selectedFunction']:checked"

    )?.value;

    if (!selectedFn || !window.uploadedFile || !window.pyContent) {

        alert("Не выбрана функция или отсутствует файл/скрипт");

        return;

    }

    const args = [];

    const fnArgs = [];

    // 1. Добавляем файл

    args.push(window.uploadedFile);

    fnArgs.push("file");

    // 2. Попробуем найти min/max вручную для order

    const minInput = document.querySelector(`#${selectedFn}\_order\_min`);

    const maxInput = document.querySelector(`#${selectedFn}\_order\_max`);

    if (minInput && maxInput) {

        const min = parseInt(minInput.value);

        const max = parseInt(maxInput.value);

        if (!isNaN(min) && !isNaN(max)) {

            args.push({ name: "order", min, max });

            fnArgs.push("order");

        } else {

            console.warn("⚠️ order min/max не заданы или некорректны");

        }

    } else {

        console.warn("⚠️ поля order\_min и order\_max не найдены в DOM");

    }

    window.currentFunction = selectedFn;

    window.currentArguments = args;

    window.argumentMetadata = fnArgs;

    console.log("✅ Аргументы переданы:", args);

    renderStep4Controls();

});

function renderStep4Controls() {

    document.getElementById("step3").classList.remove("step--active");

    document.getElementById("step4").classList.add("step--active");

    const fullContainer = document.querySelector("#step4 .data\_\_description");

    // создаём отдельные контейнеры

    fullContainer.innerHTML = `

        <div id="orderControl" style="margin-bottom: 1rem;"></div>

        <div id="chartArea"></div>

    `;

    const control = document.getElementById("orderControl");

    const chart = document.getElementById("chartArea");

    const orderParam = window.currentArguments.find(

        (arg) => typeof arg === "object" && arg.name === "order"

    );

    if (orderParam) {

        const { min, max } = orderParam;

        const label = document.createElement("label");

        label.textContent = `Параметр order:`;

        const slider = document.createElement("input");

        slider.type = "range";

        slider.min = min;

        slider.max = max;

        slider.value = min;

        slider.step = 1;

        slider.id = "orderSlider";

        const valueDisplay = document.createElement("span");

        valueDisplay.textContent = min;

        valueDisplay.style.marginLeft = "10px";

        slider.addEventListener("input", () => {

            const orderValue = parseInt(slider.value);

            valueDisplay.textContent = orderValue;

            runPythonFunction(orderValue);

        });

        control.appendChild(label);

        control.appendChild(slider);

        control.appendChild(valueDisplay);

        runPythonFunction(min); // старт с минимального значения

    } else {

        runPythonFunction(); // без order

    }

}

async function runPythonFunction(order = null) {

    console.log("▶ Вызов runPythonFunction, order:", order);

    const py = await initPyodide();

    const chart = document.getElementById("chartArea");

    if (!chart) {

        console.error("❌ Контейнер для графика не найден");

        return;

    }

    chart.innerHTML = "";

    const fileContent = await window.uploadedFile.text();

    console.log("📄 Содержимое CSV:", fileContent.slice(0, 100));

    try {

        py.FS.writeFile("input.csv", fileContent);

    } catch (err) {

        console.error("❌ Ошибка записи файла:", err);

        alert("Ошибка записи файла");

        return;

    }

    try {

        await py.loadPackage(["pandas", "scipy"]);

        await py.runPythonAsync(window.pyContent);

    } catch (err) {

        console.error("❌ Ошибка выполнения .py:", err);

        alert("Ошибка в .py-файле");

        return;

    }

    let command = "";

    if (

        window.currentFunction === "restore\_polynomial" ||

        window.currentFunction === "restore\_spline"

    ) {

        command = `${window.currentFunction}(open('input.csv', 'rb'), ${order})`;

    } else {

        command = `${window.currentFunction}(open('input.csv', 'rb'))`;

    }

    console.log("⚙ Вызов Python-функции:", command);

    let resultCsv = "";

    try {

        resultCsv = await py.runPythonAsync(command);

        console.log("✅ CSV-результат:", resultCsv.slice(0, 200));

    } catch (err) {

        console.error("🔥 Ошибка вызова Python-функции:", err);

        alert("Ошибка при выполнении Python-функции:\n" + err);

        return;

    }

    if (!resultCsv || resultCsv.trim() === "") {

        chart.innerHTML = "<p>Функция вернула пустой результат.</p>";

        return;

    }

    const blob = new Blob([resultCsv], { type: "text/csv" });

    console.log("📊 Рисуем график");

    try {

        const chart = document.getElementById("chartArea");

        chart.innerHTML = "";

        await renderChart(chart, blob); // работает так же, как для #chartContainer

    } catch (err) {

        console.error("❌ Ошибка отрисовки графика:", err);

        alert("Ошибка при отрисовке графика");

    }

}

package.json

{

  "name": "diploma",

  "version": "1.0.0",

  "main": "index.js",

  "scripts": {

    "test": "jest"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "description": "",

  "devDependencies": {

    "@eslint/js": "^9.29.0",

    "eslint": "^9.29.0",

    "globals": "^16.2.0",

    "jest": "^30.0.0",

    "jest-environment-jsdom": "^30.0.0"

  },

  "jest": {

    "testEnvironment": "jsdom"

  }

}

# Приложение 2

Применение системы на задаче обратной фильтрации изображения

Загрузка изображения

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Содержимое, созданное искусственным интеллектом, может быть неверным.

Загрузка пользовательского скрипта обработки данных

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Содержимое, созданное искусственным интеллектом, может быть неверным.

Выбор функции и параметров

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Ожидание результата

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Результат

Изображение выглядит как кот, снимок экрана, текст, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

# Приложение 3

UML-диаграмма активности системы

